

**An Information System for a
Metal-Work Company**

Jake Gibilaro

BSc Computing with artificial intelligence.

2003/2004

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)_____

SUMMARY

This project aimed to solve the information management problems faced by a small company, Gibilaro Design, that specialises in the design and manufacture of metal furniture. The company has expanded substantially in recent years and is faced with a growing amount of information regarding stock, suppliers, customers, projects and accounts. The owner and employees have been finding this information increasingly difficult to manage.

The problems faced by the company were solved by realising the following set of minimum requirements:

- Decide on a plausible solution based on analysis of the existing system and user requirements.
- Choose and follow a suitable design methodology
- Develop a new system using an appropriate technology.
- Create a user manual for the system.

A demo-version of the system built (with dummy-data and slightly limited functionality since some operations require the user to be on the same computer as the server) is available to view at:

<http://wwwdev.comp.leeds.ac.uk/ctzjsg/index.php>

(username: jaketest , password:mif8926).

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 The Problem

1.2 The Current System

- 1.2.1 Users
- 1.2.2 Information Storage
- 1.2.3 Stock
- 1.2.4 Suppliers
- 1.2.5 Agents and Customers
- 1.2.6 Projects
- 1.2.7 Accounts and Invoices
- 1.2.8 Summary of information held

1.3 Why a new system is needed

- 1.3.1 Stock Control
- 1.3.2 Cross referencing of records
- 1.3.3 New and temporary staff
- 1.3.4 Other Problems

1.4 Possible Solutions

1.5 Similar Projects

1.6 Methodology

- 1.6.1 Why follow a methodology?
- 1.6.2 The System Development Life Cycle (SDLC)
- 1.6.3 Importance of User Involvement
- 1.6.4 Prototyping
- 1.6.5 Methodology for this Project

Chapter 2: Design

2.1 New System Requirements

- 2.1.1 Stock
- 2.1.2 Customers
- 2.1.3 Suppliers
- 2.1.4 Projects
- 2.1.5 Invoices

2.2 Functional Requirements

2.3 Non-Functional Requirements

2.3.1 Usability

2.3.2 Remote Access

2.4 Possible future expansions

2.5 Choice of technology for the new System

2.5.1 Web-based Interface

2.5.2 Relational Database

2.5.3 PHP

2.5.4 MySQL

2.5.5 Apache

2.5.6 Internet Explorer

2.6 The Prototype

2.7 The Database

2.7.1 Entity-Relationship modelling

2.7.2 Mapping the ER model to a relational Scheme

2.7.3 Normalisation

2.7.4 Integrity Constraints

2.8 The User Interface

2.8.1 General Usability Issues

2.8.2 Web Perspective

2.8.3 Other Usability Issues

Chapter 3: Implementation

3.1 The Programming Environment

3.2 GET and POST requests

3.3 The Database

3.3.1 Creating the tables

3.3.2 SQL Queries

3.4 PHP Files

3.4.1 Forms

3.4.2 Add Files

3.4.3 Display Files

3.4.4 Delete Files

3.4.5 Edit Files

3.4.6 Other Files

- 3.5 Illustrated example of system in use
- 3.6 Other Technologies used
- 3.7 Testing
- 3.8 Installation

Chapter 4: Evaluation

4.1 Methodology

- 4.1.1 Choice of Methodology

4.2 Design

- 4.2.1 Analysis of the Current System
- 4.2.2 Functional Requirements
- 4.2.3 Choice of technology
- 4.2.4 Database
- 4.2.5 User Interface
- 4.2.6 The Prototype

4.3 Implementation

- 4.3.1 Coding Issues
- 4.3.2 Remote Access

4.4 Overall Success of the Project

Bibliography

Appendices

- A: Personal Reflection
- B: Key Stages In the Project
- C: Interview with Karen Gibilaro
- D: Part of Test Plan
- E: Usability Tests
- F: The User Manual

CHAPTER ONE: INTRODUCTION

This chapter provides an introduction to the project, describing Gibilaro design's current system and explaining why a new one is required. Software methodologies are discussed and the steps that were involved in carrying out this project are detailed.

1.1 The Problem

The primary aim of this project was to produce an information system for *Gibilaro Design*, a company specialising in the design and manufacture of metal furniture.

The company produces made to order goods such as furniture and fireplaces, as well as carrying out antique restorations. They have also been involved various other types of work including public art projects. The business is located in a workshop with specialist equipment and storage areas with an on-site office. Information relating to the running of the company is held on an office computer and in various filing cabinets.

The company has grown substantially in recent years and has amassed a substantial amount of information. As the company continues to expand the current system is becoming increasingly difficult to manage. The owner has considered various solutions but has yet to find a product entirely suited to the company's needs and budget.

This project aimed to solve the information management problems faced by the company by managing stock, customer, supplier, invoice and project information in an easy to use system.

1.2 The Current System

As is explained in **1.6.5** a detailed analysis of the current system was undertaken to establish the users of the system, the types of information and how it is stored and processed.

Throughout this chapter 'the system' refers to all elements of the company's running that are relevant to this project: the management of stock, supplier, customer, invoice, and project information.

The following information was gathered from two visits to Gibilaro design and several phone calls. Details of an interview with Karen Gibilaro (the office manager) that was carried out on the first of these visits can be found in Appendix C.

1.2.1 Users

The users of the system are the employees of the company. Currently there are three permanent employees as well as temporary staff who come in when needed. At present the three permanent Employees are:

- Jesse Gibilaro: The registered sole trader who oversees the general running of the company.
- Karen Gibilaro: The office manager.
- Sam Smith: Assistant technician.

While Karen Gibilaro is the chief user of the system, all employees can be considered users, as they access and update information (e.g. the Assistant Technician may perform a stock check, or record details of an invoice).

The employees have little computer experience beyond Microsoft office and basic file management in Windows operating systems. This was a significant issue when designing the system and is discussed in 2.3.1.

1.2.2 Information Storage

The company currently has one computer in the office running Windows XP. For each year there is a folder storing all company information (invoices, projects, letters and orders). A filing cabinet in the office and several shelves around the workshop store folders with information on previous projects. Received invoices are stored in ring binders. The files stored on the computer are backed up to zip-disk every few months.

1.2.3 Stock

A great many different stock items are required due to the diverse nature of the company's produce. Items include various types of sheet metal, mouldings, and smaller items such as ball bearings.

Quite often supplies are only restocked when they are noticed to be running low. Stock checks are carried out fairly regularly with information on current stock levels kept on paper temporarily in the form of handwritten notes to serve as reminders to employees. Since employees are not expected to update stock information every time an item is used, these notes tend to be inaccurate (except for immediately after a stock check).

1.2.4 Suppliers

The company deals with a large number of suppliers, both in the UK and abroad. Currently there are 21 suppliers who are used regularly. Although some suppliers are used very frequently, there are no regularly scheduled deliveries as the made-to-order nature of the company's produce means required stocks are always changing. Stocks are ordered from suppliers mainly by fax but also by telephone and occasionally over the Internet.

Information on suppliers consists of an address and phone number list held on a MS-word document on the office computer; printed copies are kept next to two phones that are outside of the office. Often numbers are added by hand to the lists; occasionally the word document is updated from the printed lists.

1.2.5 Agents and Customers

Most projects come through agents such as antique/art dealers and architects. Generally there is little correspondence with the customers themselves. Likewise payments usually come from the dealers but occasionally are direct from the customer. There are 10-15 different agents most of whom regularly supply the company with jobs.

Names and addresses of the agents and customers are stored in the same document as suppliers.

1.2.6 Projects

The company carries out around 15-20 major projects a year as well as a large number of smaller ones. Each major project is dedicated a ring-binder type folder stored on various shelves in the office containing all paper documents relating to the job (such as plans and dealer/customer correspondence). Each project is identified by either the name or address of the dealer or client or by a brief description of the job. Folders for older jobs are moved from the office and kept in one of several shelves in the workshop.

One point of note is that there is no distinction between finished projects and currently running ones within the system. It is unlikely there will be many projects running at any one time, so users will know which ones are current.

1.2.7 Accounts and Invoices

All printed invoices (e.g. orders from suppliers) are stored in a large ring binder and organised by date. Invoices to customers and agents are stored as word documents in each year's folder.

Gibilaro Design is legally obliged to hold accounts information going back 6 years for VAT purposes. This is held on an MS Excel Spreadsheet, with summaries and VAT returns generated automatically. When an invoice from a supplier arrives, its details are input into this spreadsheet, likewise when an invoice to a customer has been paid, the details are entered.

1.2.8 Summary of information held

The following table summarises the types of information held and how it is stored:

Type of information	How it is held
Stock	No permanent records
Suppliers	Name and Number list (MS Word document and printed copies)
Agents/customers	Name and Number list (MS Word document and printed copies)
Projects	Folders in a filing cabinet and on the computer.
Accounts and invoices	Ring binder for invoices from suppliers/sub-contractors. Word documents for customer invoices. MS Excel Spreadsheet for VAT purposes.

Figure 1-1: Summary of information held

1.3 Why a new system is needed

1.3.1 Stock Control

The main problem with the current system is with stock control. With no formalised method for recording stock information, supplies can run out unexpectedly. Since some stocks can take weeks to arrive (such as those ordered from abroad) this can cause projects to be delayed. Currently the company attempts to ensure this does not happening by over-ordering stock, and by having intermittent stock checks as explained in 1.2.3.

1.3.2 Cross referencing of records

As suppliers are listed alphabetically rather than by the stock they supply, sometimes it is hard to locate a particular supplier for a stock item. For some suppliers, what they supply is not recorded at all (the employees have to remember it). The office manager has knowledge of which supplier supplies which item but other employees have often had trouble, especially when the office manager is away.

An employee may want to look back at a previous project (for example to look at plans if a new project is similar to a previous one). It can be difficult to find this information, as folders are stored in different places and not always in any particular order.

Sometimes an employee needs to refer to an old invoice: for example to keep prices in line for a particular dealer in order to maintain good relations. Again this can be difficult as invoices and project folders are stored by date rather than the dealer or the project.

1.3.3 New and temporary staff

Gibilaro Design often employs temporary staff to cope with periods of increased demand. At present the running of the system is not straightforward for a newcomer to understand: permanent staff have become used to running the system, filling in any gaps themselves. Temporary staff generally do not use the system as there is not time to train them. With customer, supplier, stock and accounts information integrated into one easy-to use system a new or temporary employee could quickly gain insight into the workings of the company and start performing administrative duties.

1.3.4 Other Problems

All invoices are numbered sequentially, the only way to tell the number of the current invoice being written is to locate the last one, in the appropriate folder on the computer. Consequently invoices are sometimes incorrectly numbered.

Staff have also complained of time wasted flicking through printed lists, looking for phone numbers for suppliers.

1.4 Possible Solutions

There are a large number of stock control/information systems available. Many of these are specific to a particular sector (for example: BCP(2003) offer a stock control system for the food and drink industry). Gibilaro Design is a unique company (the only of its kind in the country) so it is especially difficult to find such a system that is suitable for its needs.

There also exist a number of general-purpose stock-control systems such as Stockit by Number One Systems(2003) which was considered as a possible solution. This is a powerful stock control system, based on an Access database. It has some useful functionality such as determining inventory shortages and automatically creating orders, it also can integrate with existing Excel spreadsheets by converting data to a file format (Comma Separated Variables - .CSV) that can be opened in Excel.

However for Gibilaro Design, Stockit offers much more functionality than is needed (for example having three different part numbers for each item depending on the supplier and individual manufacturer) – and more importantly has a complex interface that would require extensive training for the staff who have little computer experience:

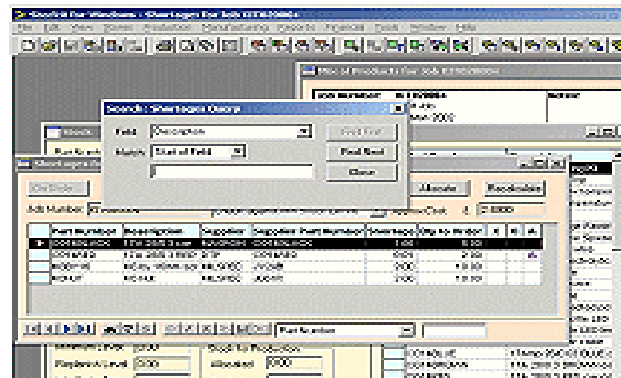


Figure 1-2: Stockit's complex interface

As for converting data to .CSV format (in order to integrate the existing accounts system - see **2.15**) this is fairly easy to achieve with most relational database systems (see **3.4.6** for how it was done in this project). Systems like Stockit also tend to be expensive whereas the system built in this project was implemented with all the required functionality using only free technology (see **2.5**).

1.5 Similar Projects

There have been many student projects in the past involving stock control and information systems, several of which were examined. The unique nature of this project does make it hard to draw many comparisons, but one project that has some similarity to this one (in that it deals with stock control) is briefly discussed.

A stock control system for a record shop was created by Johnson(2002). His system differs from this project in that it is primarily concerned with stock control, which is integrated with sales (when something is sold, it is automatically removed from stock). He chose to use Access, which can be used to implement a reasonably complicated database and user interface fairly quickly with limited technical ability. The problem with this is that it does not give the developer a lot of freedom, particularly in terms of the user interface. The interface of Johnson's system was typical of one produced by Access - with all data operations carried out using various forms. Such an interface

would have been inappropriate for this project since the users are have no experience with Access and **are generally lacking in software skills**. For this project the final system was designed to have a user friendly interface partially based on that of a website (see **2.8.2**) that attempted to provide a familiar environment for the inexperienced user, as well as functionality in the data returned rather than just the forms to make the interface more intuitive and efficient (see **2.8.3**). It is not clear how such an interface could be built using Access.

A major disadvantage is that Access is only supported by Microsoft operating systems. Although Gibilaro design is currently using Windows XP there is no guarantee that this will always be the case. For this reason, the technology used to was all cross-platform as explained in **2.5**. Access is also expensive, so it would never be an option in cases like this where superior open-source technologies for building the system are available.

Access was the predominate choice amongst the projects looked at, the same criticisms apply to all: fast implementation at the expense design freedom, the fact they are limited to Microsoft operating systems, and the high price.

1.6 Methodology

1.6.1 Why follow a methodology?

In the early days of software development the focus was largely on the programming aspect of a project, often without breaking down the project into various sub-tasks. The project was carried out largely by technical personnel, who often found it hard to communicate with the non-technically trained end users of the system. This communication problem along with a general lack of planning meant that user requirements were often overlooked and it was difficult to realistically determine completion dates (Avison and Shah, 1997).

The upshot of all this was that user needs were frequently not met and projects often ran on long past their completion dates. Even once projects were completed, programmers often had to spend a lot of time rewriting code in order to satisfy users.

A software methodology is a set of rules and guidelines to use in the development of a piece of software that aims to prevent the above problems.

1.6.2 The System Development Life Cycle (SDLC)

This structured process for developing information attempts to avoid the problems discussed above. It specifies a number of phases, all of which must be completed, for carrying out a software project.

The details of the SDLC vary across different sources, but the underlying principle is the same. The SDLC as described in *The Systems Development Life Cycle: A First Course In Information Systems* Avison and Shah(1997) consists of 6 phases as follows:

- 1) Feasibility study: Determines whether the project is feasible.
- 2) Systems Investigation: Plans the system in some detail (defines possible problem areas etc.). The current system is studied and knowledge-gathering techniques such as interviews can be used.
- 3) Systems Analysis: Detailed analysis of current system to determine the requirements of the new system.
- 4) Systems design: Determines which technologies will be used to build the system, and provides a detailed design for the new system.
- 5) Implementation: The system is built and tested.
- 6) Review and maintenance: changes are made in response to problems with the system

Traditionally, phases in the SDLC were carried out linearly - with each step being carried out in sequence. The more sensible 'fountain' approach is to cycle back through the stages if required (for example changes can be made in the design once implementation has already begun) – this is especially useful given the varying nature of user requirements.

1.6.3 Importance of User Involvement

The problems that arise when user requirements are not met can be largely avoided if the user is involved throughout the development process, and communication between users and developers is effective in that any requirements agreed on have the same meaning to both parties (Augerou and Cornford, 1998).

Software development methodologies such as Rapid Applications Development (RAD) are concerned more with user involvement than with structure: employing techniques such as joint application development workshops (intensive meetings between users and developers) to ensure the final product meets all user needs (Avison and Shaw, 1997).

1.6.4 Prototyping

Prototyping, providing the user with a partial implementation of the system, is a highly effective way of ensuring user requirements are met. Faced with a prototype, users have a better idea of what they are getting, and can give feedback to the developers.

A user-interface prototype is a prototype built to explore usability issues (Wasserman and Shewmake, 1982). Building such a prototype and presenting it to the user early in the development process, can insure usability requirements are met. Such a prototype requires none of the functionality of the system to be present, and is ideal for non-technically minded users who are not concerned with the inner workings of the system. The problem with such a prototype is that, although user-input can be explored, the lack of functionality means that there is no output for the user to respond to. Attempts have been made to resolve this by means of simulating the system output (Rosson and Carroll, 2002).

Rosson and Carroll (2002) also warn of the dangers of presenting too polished a prototype. Users may commit to early without giving adequate feedback if they think what they are seeing is a true representation of the final system. A rougher prototype encourages user input as they 'fill in the gaps' themselves, it may also encourage new requirements to be discovered as the user asks questions such as ("Why can't I do X?", "How could I do Y?" etc.). Even an approach as simple as a pen and paper often provides good results and saves a lot of time: creating a detailed prototype can be time-consuming.

1.6.5 Methodology for this Project

There are a wide range of commonly used methodologies for building information systems. One such methodology, SSADM (Structured Systems Analysis and Design Methodology) is a highly structured modular approach to the analysis and design stages of systems development. For a relatively small project such as this, such a highly structured technique such as SSDAM was not appropriate. Nor was it necessary to make use of a highly user-centric approach such as RAD as briefly discussed in **1.6.3**; although some of the advantages of such a methodology were afforded by user involvement throughout the project. Examples of direct user involvement included:

- An interview with Karen Gibilaro to help analyse the current system (15/11/2003).
- A meeting with Jesse and Karen Gibilaro to discuss requirements (08/12/2003).
- Demonstration of the prototype (06/01/2004).
- Numerous phone calls to resolve any issues that arose.

For this project the methodology was based on the SDLC employing the fountain approach. The project consisted of 5 stages as follows:

Analysis of existing system

A detailed analysis of the new system had to be carried out before the requirements of the new system could be determined. This was to establish the types of information in the system, how it is stored and the potential users of the system. The findings of this analysis are largely explained at the start of this chapter (1.2).

User Requirements

A meeting with users of the system was held to discuss the requirements of the new system. This, along with the analysis of the current system, and the use of a prototype, was used to create a set of functional requirements for the new system (see 2.2).

The level of expertise of the users in terms of software skills and general computer literacy was assessed to ensure that the final system met usability requirements. Other non-functional requirements were determined (see 2.3).

System Design:

A suitable technology (PHP/MySQL) was decided on based on the scope of the system and the requirements of the users. The reasons behind this choice are explained in 2.5.

The MySQL database was designed (following relational database design principles) and normalised. This process is detailed in 2.6

The user-interface of the system was designed, in accordance with usability requirements determined. The design of the interface, from a both a general and a web perspective along with specific usability issues is described in 2.8.

Implementation testing and installation:

A user-interface prototype was presented to the user,
The system was built and tested. This process is documented in Chapter 3.

Maintenance:

Maintenance will be carried out when required. At present there are two changes to the system being carried out (changing the invoice functionality and addressing browser compatability issues - detailed in Chapter 4).

Refer to Appendix B for dates and descriptions of key stages in the project.

CHAPTER TWO: DESIGN

This chapter details the requirements of the new system and explains the choice of technology. Design issues that were involved in creating the system are discussed. From here on ‘the system’ will refer to the new system that was implemented in this project.

2.1 New System Requirements

There are many ways of determining the requirements of a system, in some cases all requirements can be obtained from the users (in cases where they have a clear how the new system will be used). For this project, however this was not sufficient as the users had no experience of having such a system installed, and did not have a detailed idea of everything the new system should or could do. In cases like this analysis and prototyping are more appropriate ways of determining the system requirements (Augerou and Cornford, 1998).

In the end three different approaches to requirements capture were used: analysis of the existing system through interviews and observation, a meeting to discuss requirements, and use of a prototype.

A requirements specification is commonly used to finalise or ‘sign off’ a project, the users agree to it and then development begins. In the case of this project the requirements specification was in the form of a list of functional requirements presented to the user on 08/12/03. These requirements are fairly high level as they should specify what the system will do rather than how exactly it will do it (Augerou, and Cornford 1998). These requirements had to be at a level where they could be understood clearly by both myself (the developer) and the users, and have the same meaning for both parties. If requirements are too detailed and they can confuse the user, not detailed enough and important issues can be overlooked.

2.1.1 Stock

An effective method of stock control was needed. The system was required to record details of all stock items held, and how much of each one is remaining. It also had to be able to add new stock items and remove existing ones should the need arise.

Users will add and remove stock from the system as new orders come in and existing stock is used. As stock information is likely to become inaccurate as users are not obliged to update stock information every time some stock is used. The system was required to prompt users to perform a stock check once a week.

To solve the problem of stock running out, the system was required to warn when levels are becoming low, with the level at which this happens specified by the user for each stock item.

A further requirement was to show the suppliers for any stock item, so staff will no longer have difficulty locating them.

2.1.2 Customers

From here on, any party who offers Gibilaro design payment in exchange for the completion of a project is referred to as a customer. These are the only parties on whom it is necessary to hold information (e.g. if a dealer is acting on someone else's behalf, it is the dealer with whom Gibilaro design will be in contact).

The system was required to store the names and addresses of all customers, add and remove customers, as well as change details of existing customers. The system was required to display a particular customer's details, or display all customers in a printable list.

In order to be able to link projects and invoices to customers, it was required that the system be able to bring up all projects, or all invoices for a particular customer.

2.1.3 Suppliers

As with customers, the system was required to hold addresses and telephone numbers of suppliers that can be accessed, updated, and printed. The system was required to link stock items with their suppliers: making it possible to show the supplier(s) for a particular item.

2.1.4 Projects

The company holds information on projects in the form of Word documents or in a ring binder (see. **1.2.6**). This will not change but the new system was required to aid the management of this information.

For each project the system was required to store information such as: a brief project description, the customer it is for and the location of the project's ring binder.

It was required to display all projects, or just projects for a particular year. The system was required to be able to find all projects for a particular customer (solving one of the cross-referencing problems discussed in **1.3.2**). The system had to also be able to bring up the electronic folder on a particular project, so users could quickly locate a project's documents.

2.1.5 Invoices

At present the company wish to keep the existing accounts system (as described in **1.2.7**) they are, however, considering integrating it into the new system at a later date. The system was however required to convert invoice data into a format that could be opened in excel to achieve at least the first stage of this integration, making it possible to easily move data from the system to the accounts spreadsheet.

Instead of using word documents, the new system was required to store invoices as data in the system. The system was required to generate an invoice for a customer, display and print it as a letter that can be sent to the customer, as well as show all the invoices for a particular customer.

The system was not required to store any data on incoming invoices. These are still stored in ring-binders as described in **1.2.7**.

2.2 Functional Requirements

The following table contains the set of functional requirements that were agreed with the user on 09/01/04.

Stock 1.Display stock items and levels 2.Print all stock items for stock taking 3.Warn when stock levels are low 4.Add a new stock item 5.Add/Remove items from stock 6.Find the supplier(s) for any stock item 7.Perform stock check 8.Prompt to check stocks	Suppliers 9.Find a supplier by name 10.Find all suppliers for a stock item 11.Display all suppliers 12.Print all suppliers 13.Add a new supplier 14.Remove a supplier
Invoices 15.Add an invoice 16.Remove an invoice 17.Display an invoice 18.Display all summaries of invoices 19.Display all summary invoices for a customer. 20.Display all summary invoices for a date. 21.Generate an invoice for project 22.Convert invoice data to .CSV format (so it can be viewed in excel).	Customers 23.Find a customer by name 24.List all projects for a customer 25.Update customer details 26.Add a customer 27.Remove a customer 28.Display all customers 29.Print all customer details
Projects 30.List all projects for a customer 31.List all projects by year 32.Jump to the project folder 33.List all projects 34.Add a new project for a customer 35.Delete a project 36.Edit a projects information	Other 37.Backup the database 38.Restore the database from a previous backup 39.Open the user manual.

Figure 2-1 Functional Requirements

2.3 Non-Functional Requirements

While functional requirements express everything that the system should do, non-functional requirements express certain constraints on this behaviour. Factors such as performance or technical constraints (e.g. running on certain hardware) are common examples of non-functional requirements (Augerou and Cornford, 1998).

There was a technical constraint in that the system had to run on the existing PC. For this project, performance constraints were not considered to be a major issue given its small scale: it was safe to assume that the chosen technology could easily handle a small, single-user database system.

Two further functional requirements are detailed here.

2.3.1 Usability

The level of expertise of the users in terms of software skills and general computer literacy was assessed by means of an informal interview to ensure that the final system met usability requirements. It was found that the experience of all the potential users is limited to commonly used office software such as word and Excel, basic file management, and web browsing.

The company do not wish to spend any great length of time training users of the system so the the system had to be simple enough for inexperienced users to learn as quickly as possible. As mentioned in **1.3.3** the new system should be clear enough to quickly give a new employee an idea of how the company is run.

2.3.2 Remote access

Jesse Gibilaro, is often away on business trips, and expressed some interest in eventually making the system accessible remotely so he could keep up to date with the running of the system, being aware of any low stock levels and unpaid invoices. A password system was required to prevent unauthorised users gaining access to the system. Users wanted a system whereby the password could be 'remembered' so it would not have to be input every time. To see how the password system was implemented see **3.4.6**.

The password system means that it will be possible to make the system remotely accessible safely. When backing up the database, the .SQL file is stored on the client machine rather than on the server

CSV files created are also stored is also stored on the client machine. There are still a number of changes that would have to be implemented before remote access will be realised, these are discussed in 4.3.2.

2.4 Possible future expansions

As mentioned above, the system may eventually be expanded to be accessible remotely with all functionality. Currently the system goes part way to integrating the current account system (converting data to .CSV format), however in the future work may be carried out to automatically update accounts (fully integrating accounts information with the system).

2.5 Choice of technology for the new System

The solution that was decided on was a web-based interface backed by a relational database. A PHP/MySQL approach using an Apache web server was the chosen technology. The browser that will be used to access the system is Microsoft Internet Explorer 6. The reasons for these choices are discussed here.

2.5.1 Web-based Interface

A web-based interface can be accessed by a standard web-browser such as Netscape or Internet explorer with the logic behind the application residing on the web-server. The reasons for choosing a web-based interface are as follows:

Familiar Environment: The first reason for choosing a web-based interface is that it provides a familiar environment for the inexperienced users. For example, when faced with a menu-bar, users will know from previous Internet experience that clicking different menu items will cause different pages to be displayed. Although browsing the Internet is different from using an information system, it is likely that even casual Internet users have come into contact with dynamic web pages where forms are filled in and information is displayed accordingly (e.g. a search engine). Also in terms of usability the web-based approach is advantageous in that standards have emerged for web design, and there is a vast amount of literature on the subject. Spool et al (1990) is an example of one such text that was used extensively in the design of the system (see 2.8.2) for details.

Rapid Implementation: This project has a relatively small time scale, where the implementation stage should take no longer than 3 months. Web pages can be written reasonably quickly, and new functionality can be easily added if required: this scalability is of great help in cases where extra requirements emerge at a late stage in development (for example as a result of user feedback from the prototype).

Prototyping: As for prototyping, this can be achieved extremely quickly in html by producing a detailed model of the system that has no functionality (not linked to a database). This approach diminishes the trade-off between detail and time-consumption as outlined in Rosson and Carroll(2002) and discussed in 1.6.4. Once the prototype has been created, adding the functionality is fairly straightforward providing the database has been implemented; this makes possible parallel implementation of the interface and the database. If the user is not satisfied with the prototype, changes can be made quickly and without undue effort.

Built in functions: Most browsers have a built in print function. The system will require data to be printed (e.g. the list of suppliers), having this functionality built in saves coding time. Also the system had to bring up a project's folder that is stored on the computer's hard disk. With html, linking to internal files is trivial: with most browsers on most operating systems it is possible to open a folder just by treating it as a standard hyperlink (e.g. link to "file://c:/myfolder").

Remote access: As for the possibility of remote access, a web-browser offers the ideal solution. The system could be run on central server and accessed remotely by a web browser – there is no client software to distribute. With any remotely accessible system there will be a security concern, however technologies such as HTTPS exist for securing web pages (see 4.3.2 for a brief discussion of how this technology could be used in securing the system).

2.5.2 Relational Database

The system clearly requires persistent data (for example customer records must remain when the system is shut down).

Data could be written to files: this approach was widely used in the early days of systems development but it lead to a number of problems. The application must parse the file to extract meaningful data, this means that a data files were specific to a particular application: changing the file

structure would mean having to rewrite the application. This led to mass duplication of data, as different applications required different types of file. As there was no standard method for creating data-files, generalised querying was impossible (Ritchie, 2002).

Database Management Systems (DBMS) provided a solution to these problems. In a DBMS, information on how the data is read (the scheme) is stored along with the data, rather than at the application level. Applications interface with the DBMS rather than the data files themselves, so files do not have to be application specific.

The defining properties of a database as given by Ritchie(2002) are:

- 1) It holds data as an integrated system of records
- 2) It contains self describing records

1) Means that all relevant data is available to the application (for this system data on customers, stocks, suppliers, invoices, and projects is held) and that relationships between data can be utilised (e.g. it is possible to link customers with their invoices). 2) refers to the schemes described above (information on how the data is to be interpreted).

Early attempts at database design attempted to provide this interrelation of data using a hierarchical structure: however this approach proved to be flawed in this structure meant that querying was complicated and often programs specific to the type of data had to be used.

The relational model proposed by Codd (1970) is an extremely flexible model that has become the principal approach for commercial data processing applications. Data is stored in two dimensional tables, making it clear to understand. For example in this system the customer information will be stored on a table something like this.

Cust_id	Name	Address	Phone number
123	Jake	Somewhere	0113 000 0000

Figure 2-2 An example Table

The cust_id field, has a unique value for each row and can therefore uniquely identify the row (fields of this type are referred to as Primary Keys). Primary keys provide a simple and effective way of linking tables. For example the system will store a table of invoices, if each invoice also stores the

value of the `cust_id` for the customer it belongs to, it will be possible to display the customer information for an invoice by means of a simple query. The `cust_id` field in the invoice table is referred to as a foreign key.

SQL (structured query language) is a standard (ISO and ANSI) language accessing data in a relational database. All data operations within the system are carried out using SQL queries.

2.5.3 PHP

Having established that a web-based interface backed by a relational database met all the needs of the new system, a choice of technology for implementing this had to be decided on.

To link the user interface to the database, a server side scripting language was required. The user's browser sends requests to the server, the server processes these requests: html pages are generated (e.g. the results of a database query) and sent back to the user.

PHP was the obvious choice for the following reasons:

Database integration: PHP is able to integrate with most databases. It is particularly straightforward to integrate with MySQL (the most popular choice, and the DBMS chosen for this project).

Ease of use: Even programmers without previous experience can quickly begin writing dynamic web pages in PHP (Ullman L, 2003). This ease of use speeds up the implementation process leaving more time for the equally important design stage of the project, making it an ideal choice for a relatively small project such as this. This ease of use comes at no cost to functionality – PHP's advanced features give it functionality far beyond the needs of this project. Unlike traditional cgi scripting technology (e.g. Perl scripts), which require separate files for html and scripts, PHP is embedded within the html files. This means that html code does not have to appear in PHP *print* statements, making it much clearer to read (Stobart, 1996).

Cross Platform: PHP is a cross platform technology: it will run on Windows, Unix, and Macintosh systems. While the system will currently be running on Windows XP, it would not have to be completely rewritten should the operating system change.

Cost: PHP is completely free. This makes it preferable over ASP (Microsoft's alternative server side scripting language).

Popularity: The above advantages of PHP are perhaps best demonstrated by its popularity. The following graph shows the exponential growth in PHP use since 1999 (this data comes from a survey carried out by Netcraft, 2003):

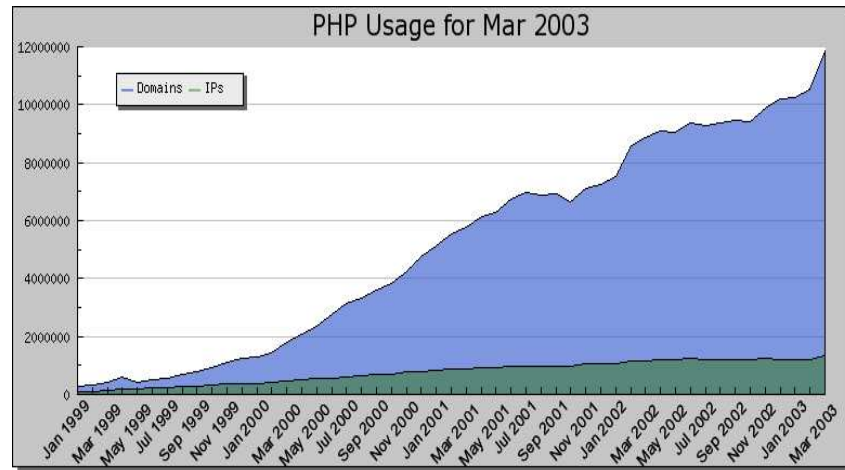


Figure 2-3 PHP Usage

Currently there are around 12 million domains running PHP, it has long since overtaken ASP as the most popular server side scripting language. In fact more than 50% of all apache web servers surveyed by Netcraft were running PHP.

2.5.4 MySQL

MySQL is the most popular open source database available and has all the necessary functionality for this project, and will have no performance issues with a small single user database. MySQL is a fairly small and compact database system that does not support advanced features such as enforcing certain integrity constraints (e.g. Foreign Key constraint – see 2.7.4). More complex RDMS such as PostgreSQL have built in functionality for this: in fact PostgreSQL was considered as a possible technology for this reason. However these constraints can be enforced at the application level as discussed in 3.3.

2.5.5 Apache

Deciding on a web server was not a difficult task. Apache has long since been the most popular web-server on the Internet: Netcraft(2003) found that more than 64% of web sites are using Apache. The reason for this popularity is that Apache is widely regarded as being the most stable, and feature-packed web server available. Its open-source development means that any bugs found are quickly repaired so Apache grows more stable with every release (Kamthan, 1999). Also the open source approach entails another advantage in that Apache is free. Apache also has full support for PHP.

2.5.6 Internet Explorer

Internet explorer was used simply because it was the only browser installed on the office computer and no advantage of using an alternative browser could be seen. However, designing the system with only one browser in mind proved to be ill advised as the final system has browser compatibility problems as explained in 4.3.2.

2.6 The Prototype

The user-interface prototype was a set of static html pages representing the layout of the final system. The prototype was left deliberately rough for the reasons outlined in (1.6.4). The following is a screenshot of the customer page of the prototype:

The screenshot shows a web page with a navigation bar at the top containing links: [STOCKS](#), CUSTOMERS, [SUPPLIERS](#), [PROJECTS](#), and [INVOICES](#). The main content area is divided into two sections. On the left, under the heading Display Customers, there is a 'Select from list:' dropdown menu showing 'Offerton, Fred' with a downward arrow, a 'Display' button below it, a 'Display All customers' button, and a 'Printable List' button. On the right, under the heading Add a customer, there is a form with the following fields: 'First Name:' (text box), 'Last Name*:' (text box), 'Address1*:' (text box), 'Address2:' (text box), 'Post Code*:' (text box), 'Telephone*:' (text box), 'Fax:' (text box), and 'Email:' (text box). Below the form fields is a note '(*required fields)' and an 'Add' button.

Figure 2-4: The Prototype

Output was simulated in the form of hard-coded html. For example hitting a search button returned data in the same format as the final system but not depending on contents of the search field itself.

The prototype was presented to the users on (02/03/04) and led to the following additions to the system specification:

- Using a text input as well as drop down menus to speed up searching for customers/suppliers.
- Only displaying names and telephone numbers for the printable lists of suppliers/customers (The list with all details can also be printed if required).
- A single name entry for customers rather than first name and second name as some customers are business not individuals.
- An extra user requirement ('Generate an Invoice for a project') was discovered.
- The colour-scheme was chosen.
- A link to an electronic version of the user manual. This was implemented as an html page that used hyperlinks to ease navigation.

2.7 The Database

2.7.1 Entity-Relationship modelling

As discussed in Chapter 4: a relational database will form the backbone of the system.

The database was designed using the Entity-Relationship (ER) model: a widely used high-level conceptual data model originally proposed by Chen P(1976).

ER modelling allows the database to be represented at a high level of abstraction and defines a set of rules for translating this model into a relational database scheme. Note that the model is general-purpose: it would be the same if any other RDMS than MySQL were used.

A model of the database was created would allow all the functional requirements detailed in 2.2 to be met. The model was then mapped to a relational database.

The database is modelled by entities (rectangles), attributes (ovals), and relations (triangles). For example the 'supplier' entity is related to the 'stock' entity by the 'supplies' relation. Attributes of 'supplier' include 'name' and 'phone number'. ER modelling can also capture the degree of a relationship: in this case a customer can have MANY (shown by *) invoices but an invoice belongs to ONE (1) customer. The primary key of each entity is the attribute which is underlined.

Participation is represented by double lines: a stock item must have a supplier, and each invoice and project an associated customer. The way in which this was enforced is discussed in 3.3.1. The final ER model for the database is as follows:

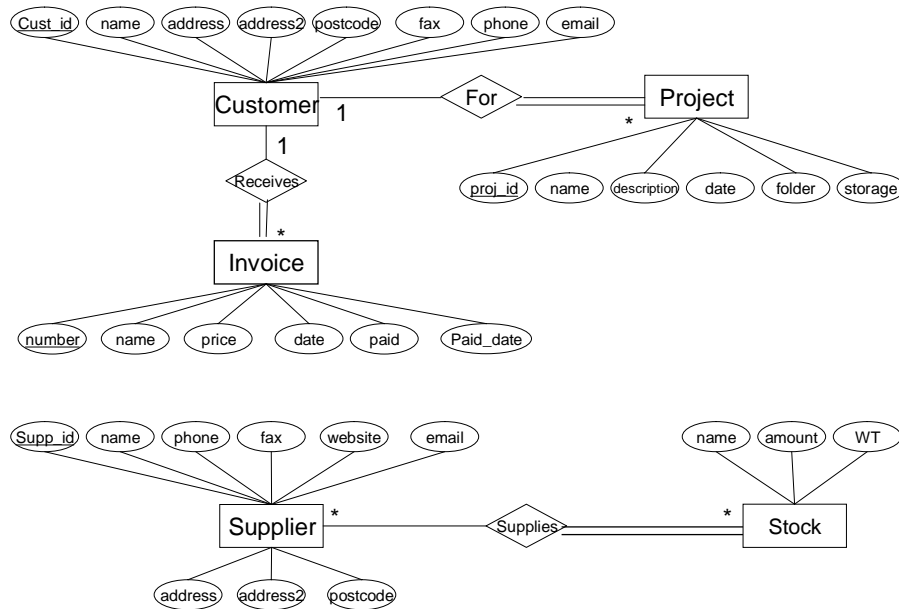


Figure 2-5: Final ER Model

Even at this high level of abstraction, the database can be shown to allow the functional requirements detailed in 2.2 to be realised. For example: it is possible to find all suppliers for a particular stock item (requirement 10) since stock items are linked to suppliers through the ‘supplies’ relationship.

2.7.2 Mapping the ER model to a relational scheme

The ER to Relational Mapping Algorithm as given by Elmasri and Navathe(2000) is a set of rules for mapping an ER model to a relational schema. It was used to as follows:

Step 1: Firstly for all the entity types, a relation is created containing its attributes. Relations: ‘customer’, ‘project’, ‘supplier’, ‘invoice’ and ‘stock’ are created with their corresponding attributes.

Step 2: The 1-MANY relationship ‘receives’ is represented by storing the foreign key cust_id (PK of ‘customer in the ‘invoice’ table).

Step 3: The 1-MANY relationship ‘for’ is represented by ‘project’ containing the containing the FK ‘cust_id’.

Step 4: For the MANY-MANY relationships, a separate relation must be created. In this case relation ‘supplies’ is created. ‘Supplies’ contains as foreign keys, the PKs of ‘stock_item’ and ‘supplier’ (stock_id, supp_id).

The final scheme for the database is shown in the following table:

customer(<u>cust_id</u> , name, address, address2, phone, postcode, fax, email)	This table stores customer details.
project(<u>proj_id</u> , cust_id, name, description, date, folder, storage)	This table stores project details – each project is linked to its customer via the FK cust_id.
invoice(<u>number</u> , cust_id, name, description, date, price, paid, paid_date)	This table stores invoice details, each invoice is linked to its customer through the FK cust_id.
supplier(<u>supp_id</u> , name, address, address2, postcode, phone, fax, email, website)	This table stores supplier details.
stock(stock_id, name quantity, wt)	This table stores stock details and how many of each item is remaining. The ‘wt’ attribute is the level which the quantity of an item must fall below to cause the system to issue a warning.
supply_stock(stock_id, supp_id)	This table represents the many-many relationship between stock and suppliers.

Figure 2-6: Database scheme

2.7.3 Normalisation

While the relational model is powerful and flexible, a poorly designed scheme can lead to problems. Consider the case where customer and project information are stored in a single table as follows:

cust_id	c_name	address	proj_id	p_name
001	Bob	Leeds	001	ProjectA
001	Bob	Leeds	002	ProjectB

Figure 2-7: A poorly designed Table

The PK of this relation has to be (cust_id, proj_id) as this pair of attributes is the uniquely determine all the other attributes of a row.

Customer 'Bob' has two separate projects (ProjectA and ProjectB). In order to record this, the fact that Bob lives in Leeds (and any other data relating to Bob) must be stored twice. This redundancy, as well as being wasteful, can lead to the following anomalies:

Imagine a new customer has to be added, but a project for them has not commenced. Since proj_id is part of the PK it will be impossible to insert a customer with no project as PKs cannot be null (see **2.7.4**). This is known as the insert anomaly.

A second problem is that if a project were deleted, all information on the customer would be lost (provided they only had one project). This is the delete anomaly.

Finally, imagine a customer changes address (or some other detail changes) if this information was stored in multiple tuples (as in figure **2-7**) two updates would be required – it is possible that one could be overlooked, leading to inconsistent data. This is known as the update anomaly.

These problems could prove disastrous for an organisation, with data becoming inconsistent, inserts failing and records disappearing. They must be avoided at all costs.

Database normalisation (originally proposed Codd, 1972) is a process of restructuring databases in a way that makes such anomalies impossible. It will generally result in tables such as figure **2-7** being broken down into smaller tables. A condition named Boyce-Codd Normal Form (BCNF) has been defined which guarantees the three anomalies described above cannot occur (Elmarsri and Navathe, 2000).

BCNF uses the concept of functional dependency (FD). One set of attributes (A1,...AN) of a relation is said to functionally determine another attribute P of the relation in the case where: if two rows with have the same values for (A1...AN) must have the same value of P. For example in the Customer relation in Figure **2-7**, cust_id functionally determines c_name and address etc. (written: cust_id → c_name, address). Two tuples with an identical customer_id would refer to the same customer.

For complex databases, it is not trivial to determine functional dependencies and rules exist for inferring new FDs from existing ones (Elmarsri and Navathe, 2000). In the case of the database in this project that uses small tables and artificial keys – it is clear what the FDs are.

Trivial dependencies are obvious cases such as: $\text{cust_id} \rightarrow \text{cust_id}$ or $\text{cust_id, name} \rightarrow \text{cust_id}$ (where an attribute is determining itself).

For a relation R to be in BCNF the following must hold: when there is a non-trivial dependency $(A_1 \dots A_N) \rightarrow R$ then $(A_1 \dots A_N)$ is a superkey of R. A superkey being a set that contains a key.

The customer relation is BCNF:

$\text{cust_id} \rightarrow \text{name, address, phone, email.}$ (the only non-trivial FD)
 cust_id is a key (thus a super-key).

Whereas the figure 2-7 is not.:

$\text{proj_id} \rightarrow \text{p_name}$
 proj_id is not a superkey.

All the relations in the database conform to BCNF in a similar way to customer (one key which defines all the other attributes being the only non-trivial FD), thus none of the above mentioned anomalies could arise.

2.7.4 Integrity Constraints

A primary key uniquely identifies a tuple of a table; therefore every primary key in a table must be unique and not null. This is known as the Entity integrity rule.

A foreign key is used to link one table to another; therefore a foreign key in a table must correspond to a primary key in the table it is referencing. For example if the project table has a cust_id value of 012, 012 must exist as a primary key in the customer table. This is known as the referential integrity rule. The way in which these constraints were enforced in the system is described in chapter 3 (3.3.1 and 3.4.1).

2.8 The User Interface

Usability was a key concern given the computer inexperience of the users. This section discusses usability issues considered when designing the user interface of the system. Consult the user manual (Appendix B) for a detailed impression of the final layout of the interface.

2.8.1 General Usability Issues

One of the most popular and widely used set of guidelines for producing a usable system are the Usability Heuristics proposed by Jakob Nielsen. Originally proposed in 1990, and somewhat refined since, the latest version can be found online at Nielsen(2004). These guidelines were used extensively when designing the user interface of the system, several of the Heuristics that proved particularly relevant to this project are discussed here.

Match between the system and the real world: This heuristic basically states that the system should use words and phrases familiar to the user rather than using system-oriented terms. This was particularly important in this project given the inexperience of the users. All text in the system is in plain English: e.g. “A new supplier has been added” rather than “New data added to the database”. Error messages are also in plain English.

Visibility of system status: This relates to the system giving appropriate feedback to the user. The system provides details of exactly what has happened as a result of a user’s actions: E.g. Adding a customer will display the message “A new customer has been added, details for (name) added to the database”.

User control and freedom: This relates to a user being able to quickly leave a state that they have entered accidentally. This was achieved in the system by extensive use of ‘close’ and ‘back’ buttons. E.g. If a user accidentally clicks ‘remove’ instead of ‘add’ for a stock item, they can simply press ‘back’ to return to the item list. Any operations that remove data from the system prompt the user with an “Are you sure” type message.

Recognition rather than recall: This relates to the functionality of the system being obvious to the user: they do not have to remember how it works. An important issue again as minimising user training was key to the system; this was achieved by consistency (see 2.8.3) and by having clear instructions throughout - forms and buttons are clearly titled (e.g. “Add Customer” or “Display all Suppliers”).

Error Prevention: This states it is better to prevent errors in the first place than have to report them. Extensive use of drop-down menus made it harder for users to make input errors.

Help and documentation: This heuristic states that although it is better if users can use a system without any help, it is sometimes better to provide it. Although the system is simple to use, the users are very inexperienced so help is provided in the form of the 'site map' (see 2.8.2) and the user manual. The user manual can be found in **Appendix F**.

2.8.2 Web Perspective

As mentioned in 2.5.1 there is a huge amount of literature on web design. Spool et al (1999) was found to be the most useful example as it focused on data from real sites rather than just theory. This book provided a number of ideas that were used to make the system as usable as possible; some of these ideas are discussed here.

Navigation bar: The system has a navigation bar to move between the different sections. This was placed at the top of the screen rather than at the side to prevent users from 'scrolling' it off the screen. The navigation bar can be seen on the screenshot below (figure 2-8).

Starting Point and Site Map: Web pages with a familiar starting point prove easier to navigate. The system has such a starting point which also serves as a minimal user manual. This idea was inspired by a 'site map': a device commonly used by web designers. These have been shown to increase the usability of websites by helping users quickly learn how to navigate sites. The following picture shows the 'site map' section of the starting page.

Stocks	Customers	Suppliers	Projects	Invoices	?
View, add and remove stocks. Print out a form for a stock check. Display any stock warnings.	Find and display customers. Show all customers. Add/Remove customers. Edit customer info.	Find and display suppliers. Show all suppliers. Find a supplier for a stock item. Add/Remove suppliers. Edit supplier info.	Find and display projects. Show all projects. Find a project for a customer. Add/Remove projects. Edit project info.	List invoices for a customer or year. List all invoices. Display an invoice (to print). Add/Remove invoices. Paid invoices→excel.	

Figure 2-8: The 'Site Map'

White space and scrolling: White space should generally be avoided: there is a limited amount of screen space available and wasting any will make users have to work harder in terms of scrolling and additional navigation. The system has multiple forms on each page so no screen space is wasted and little navigation is required (see 2.8.3). Scrolling should also be avoided as much as possible as users are not always aware that they can scroll and may miss out entire sections of the site. The system requires no scrolling when on the form pages.

Feedback: Users often have problems when it not obvious that something has changed (e.g. as a result of clicking a button). The system uses separate windows for forms and data making it obvious that a change has occurred (a data window will pop up as a result of clicking a display button).

Graphics: Using unnecessary graphical elements has been shown to have a negative effect on usability as they tend to confuse users and draw attention away from functional elements of the site. While the system does have colourful buttons for aesthetic reasons, non-functional graphical elements are avoided.

2.8.3 Other Usability Issues

Consistency: Each page of the system has a form on the left for displaying data and one on the right for adding data and data is always displayed in a new window. Similar operations are carried out in similar ways – e.g. the process for deleting a customer is the same as for deleting a supplier. Consistent use of colour is also used (e.g. delete buttons are always red). The following screenshot is of one of the form pages (stocks.php):

The screenshot shows a web application interface for managing stocks. At the top, there is a navigation bar with tabs for 'Stocks', 'Customers', 'Suppliers', 'Projects', and 'Invoices'. A help icon (?) is located on the right side of the navigation bar. The main content area is divided into two columns. The left column contains a form for 'Add/remove stock' with a text input for 'Item Name', a 'Display' button, and a dropdown menu for 'Or choose from list' with a 'Display' button. Below this is a 'Display All Items' button. The right column contains a form for 'Add a new stock item' with a text input for 'Item Name*', a dropdown menu for 'Supplier*' (showing 'Barry's Bolts'), and a text input for 'Warn when only*' with an 'Add' button. At the bottom of the page, there are two buttons: 'Printable Stock Check Form' and 'Perform Stock Check'.

Figure 2-9: Stocks form

Two Ways of editing stocks: Stock levels are updated in two ways, either immediately following use of stock or by a stock check (see 2.1.1). The system reflected this: it was possible to bring up all items and edit levels, or add/remove stock for an individual item. The idea is that the user uses the system to print a form for a stock check, use it to carry out the check, then use the system to bring up all items and edit accordingly.

Interactive Data: The data returned is interactive: it has buttons to perform operations on the data returned e.g. 'delete', 'edit' or 'send email'. This prevents the initial forms from being cluttered, is intuitive, and ensures users know exactly what data they are operating on. This also adds some redundancy to the interface: a user could bring up a project from the projects page or by clicking the 'projects' button for a user. (see figure 2-10 for an example of some data returned).

Drop down menus: Although validation means that invalid data cannot be entered into the database, a user making a typing error could fail to display the data they wanted. Using drop down menus means mistakes like this cannot be made, saving the user some frustration.

Striped Data: Data returned is striped – alternate rows are white and grey. This makes it clear for a user to see what row they are looking at: this is particularly important in the system as rows have buttons such as 'delete'. The following screenshot shows project data displayed by the system:

Name	Customer	Description	Date	Folder	Storage	
12 candlesticks	Testerson, Fred	Made 11 novelty candlesticks and an egg cup	02/01/2003		upstairs top shelf	Edit Delete
Big fireplace	Gibilaro, Jake	Installed a great big fireplace in the living room.	01/02/2004		downstairs main filing cabinet	Edit Delete
Little fireplace	Gibilaro, Jake	little fireplace in the basement	01/02/2000		None	Edit Delete
Stair Case	Offerton, James	Fitted a steel staircase	03/02/2003		Upstairs filing cabinet	Edit Delete

Figure 2-10: Project Data

CHAPTER THREE: IMPLEMENTATION

This chapter explains key idea involved in the implementation of the system. It is not the intention of this chapter to detail every aspect of the implementation process, but to give a general overview of how the system works. Testing and installation is also documented here.

3.1 The Programming Environment

The system was built on a desktop computer running Windows XP. Before implementation could begin, Apache, PHP and MySQL all had to be installed and correctly configured. This was achieved almost effortlessly thanks to phpdev, a program that automatically installs and configures Apache, PHP, and MySQL on any Win32 machine. Phpdev is available from firepages (2001).

Phpdev also installs a program called phpmyadmin: an easy to use web-based user interface (written in PHP) for managing MySQL databases. This interface proved useful when testing the system (e.g. deleting unwanted records before delete functionality of the system was implemented), and was also used for testing all SQL queries before writing them into the system.

3.2 GET and POST requests

There are two methods available for sending data via HTTP from the user's browser to the server for processing.

The GET method sends name/value pairs to the server by appending them to the URL. In the system, buttons are created on the display pages using data from the database as the values appended to a URL. For example a typical delete button on the display customer page would be a hyperlink to: `deletecustomer.php?cust_id=120`.

Any variables sent by a GET request can be retrieved in PHP by accessing what is called the `$_GET` array as follows:

```
$myvariable = $_GET['name'];
```

The POST method sends name/value pairs within the body of an HTTP request. It is used in the system where data is being sent from forms (e.g. displaying a customer by name). These variables are accessed in a similar way to those sent using the GET method, this time by accessing the `$_POST` array.

3.3 The Database

3.3.1 Creating the tables

Tables were created using SQL queries. For example, the customer table was created with the following query:

```
CREATE TABLE customer (  
    cust_id bigint(20) unsigned NOT NULL auto_increment,  
    Name varchar(30) NOT NULL,  
    address varchar(40) NOT NULL,  
    address2 varchar(40) NOT NULL,  
    postcode varchar(40) NOT NULL,  
    phone varchar(20) NOT NULL,  
    fax varchar(20) NOT NULL,  
    email varchar(30) NOT NULL,  
    PRIMARY KEY (cust_id)  
);
```

Input validation means the system never allows a null value to be entered into the database. However, even at the database level, fields were not allowed to be null in order to facilitate the testing process. nulls should generally be avoided as they can give rise to unexpected results (Elmasri and Navathe, 2000).

Entity Integrity (as described in **2.7.4**) was enforced using MySQL's `auto_increment` function (a new entry will have a primary key of `n+1` where `n` is the primary key of the last entry).

3.3.2 SQL Queries

As explained in 2.7.1 All operations on data were carried out using SQL queries. Queries were tested against the database before being coded into the system. For example the following query is used to find a supplier for a stock item:

```
SELECT su.* FROM stock s INNER JOIN  
supply_stock ss ON s.stock_id = ss.stock_id  
inner join supplier su ON su.supp_id = ss.supp_id WHERE s.name = 'bolt1A'
```

In this query, the stock table is joined to the supplier table through the many-many table supply_stock. This query will return supplier information for all the suppliers who stock the item 'bolt1A'.

The system uses various different types of SQL queries:

SELECT queries: Used for retrieving database records

INSERT queries: Used for adding records

UPDATE queries: Used for updating existing records

DELETE queries: Used to delete records

3.4 PHP Files

3.4.1 Forms

The files *stock.php*, *customers.php*, *invoices.php*, *suppliers.php* and *projects.php* contain forms for adding and displaying data. These forms use the POST method to send user input to the add and display files. For example on *customers.php*, a user clicking the Add button on the 'Add a customer' form will cause all the information in this form to be posted to *addcustomer.php*. *Invoices.php* has an additional form for operations regarding paid invoices, and *stocks.php* has a form for stock checks as well as functionality that warns if a stock item is running low (using a SELECT query to see if there are any items with quantity < WT).

Invoices.php also warns the user of any invoices that have not been paid for a certain time. The PHP *getdate* function is used to return today's date in the form MM/DD/YYYY. This can then be converted to a UNIX timestamp(the number of seconds since January 1st, 1970) using the PHP function *strtotime*. For each unpaid invoice, its date is converted into a timestamp and subtracted from

the current day's timestamp. Dividing by 86400 (the number of seconds in a day) returns the number of days; if this is greater than 30 JavaScript is generated to display a warning in new window (see **3.3.7 warninv.php**). As dates in the invoice table are in the form DD/MM/YYYY a function was written to convert them to DD/MM/YYYY using PHP substring functions so timestamps could be created.

Participation and referential integrity explained in **2.7.4** are enforced in these forms. For example to ensure that a project must have an associated customer, a customer must be chosen from a drop-down menu when adding a project. An example of a form page is shown in the screenshot figure **2-9**

3.4.2 Add Files

The files *addcustomer.php*, *additem.php*, *addinvoice.php*, *addsupplier.php* and *addproject.php* receive user input from the forms as explained above. This input is validated before being entered into the database (for example NULL values are never allowed and the date entered for a project must be valid). If the input is invalid, an error message is displayed and no data is entered into the database. If the data is valid it is added to the appropriate table by means of an SQL INSERT query. In the case of adding a new stock item, more than one table will have to be updated – the stock_id of the new item along with the supp_id of the supplier who supplies it must be inserted into the supply_stock table as well as inserting details of the new stock item into the stock table.

3.4.3 Display Files

These files (*displaycustomers.php*, *displayinvoice.php*, *displaysuppliers.php*, *displaystock.php*, and *displayprojects.php*) receive data from the forms as explained above. For example pressing the 'Display' button will post a two variables to *displaycustomers.php*: one causing it to issue an appropriate SELECT query, and the other containing the user's input from a text area or a drop down menu which will be used in the WHERE clause of the query. Data is displayed using functions, in order to reuse code (e.g. displaying all customers, or one from a text input, or a drop down menu all require the same type of data to be displayed). Some files have more than one – for example *displaycustomers.php* has a function to display all customer details and one to just display names and phone numbers for printing. This function loops through every row returned by a query and returns the appropriate data, creating hyperlinks where required (see below). These files will always be opened in a new browser window. An example of data being displayed by the system is shown in figure **2-10**.

As explained in **2.8.3** the data returned by these files is interactive. For example displaying customer information will return a table that has buttons to delete and edit that customer's details as well as a button that will take the user directly to any projects for that customer. These buttons were implemented as hyperlinks: for example the button to display the customer's project is a hyperlink to:

```
Displayprojects.php?cust_id=n
```

This is using the GET method to send the `cust_id` to *displayprojects.php*, which will return the projects for this customer using the following SELECT query with the `cust_id` in the WHERE clause:

```
SELECT customer.Name, project.* FROM
project, customer WHERE project.cust_id = customer.cust_id
and customer.cust_id='n' order by project.name
```

All buttons on the returned data are implemented in this way.

3.4.4 Delete Files

The files *deletecust.php*, *deleteinv.php*, *deleteitem.php*, *deleteproj.php* and *deletesupp.php* remove data from the database. These files receive input in the form of GET requests from the display files as explained above. The data received is the primary key of the tuple that is to be deleted. If the user confirms, the appropriate record is removed from the database by means of a DELETE query.

To preserve referential Integrity (see **2.7.4**) often multiple delete queries are required. For example deleting a customer requires that all projects for that customer be deleted. Deleting a supplier is complicated by the many-many relationship between stock and suppliers. The system must check if this deletion has resulted in a stock item existing that has no suppliers, and if so delete that stock item.

3.4.5 Edit Files

These files (*editcust.php*, *editproj.php*, *editsupp.php*) again receive input from the display files in the form of the primary key of the tuple to be edited by means of a GET request. These files use this key to select all attributes of the tuple using an SQL query and display them in a form that can be edited. Once the user has made any changes, they click an 'update' button that will post the new data back to the same edit file, this data is then validated and inserted into the database by an UPDATE query.

3.4.6 Other Files

authenticate.php: When running on Apache, PHP supports HTTP authentication. Using the `PHP header` function (this function sends an http header to the clients browser, redirecting it to a page, file or in this case a password prompt), the user is prompted for a username and password. The input is then checked against a table of usernames and passwords in a database using a `SELECT` query as follows:

```
select user from users where user = '{$_SERVER['PHP_AUTH_USER']}' and  
password = PASSWORD('{$_SERVER['PHP_AUTH_PW']}')
```

The `$_SERVER` variables are the user input, and the MySQL `PASSWORD` function is used to encrypt the password before comparing it to the table (passwords are stored in an encrypted form for security reasons). If this query is successful (the name and password match a tuple in the database), a variable is set that is checked at the start of every file to ensure the user is authorised. Passwords can be remembered by the web browser on the user's password list. The password system required an extra table ('users') to be created, that stored user names and encrypted passwords. The following screenshot shows the user being prompted for a password that has been 'remembered':



Figure 3-1: HTTP authentication.

index.php: The index page includes PHP code that prompts for a stock check to be taken once a week, using a PHP function to return today's date. It has a link to an electronic version of the user manual, and links to the files for backing up and restoring the database.

dump.php: this is used to backup the database by dumping it to a .sql file (which contains all the SQL queries needed to build the database and add all the data). It uses code adapted from a script posted on the Internet by Keyur Itchhaporla at 1phpstreet(2004).

The filename is given as the current date e.g. 11-12-2004.sql. This file is then sent to the client machine using a PHP header function, it is downloaded by the browser with the user specifying where to save it:



Figure 3-2: Saving an SQL file

backup.php: Again this file includes code adapted from a freely available online script (this time posted on the forums of devarticles.com(2004) by Mike_r). Clicking the 'restore' button on the index page brings up this page in a separate window. The user can then browse for an .SQL file from which to backup the database.

Validation is carried out by checking that the extension of the file is .sql and that the first line of the file contains the string 'GibilaroDumpFile' which was inserted by dump.php. If the file is valid, backup.php uses it to back up the database by simply running all the queries against the database.

mysqlexport.php: This file converts paid invoices to .CSV format. The user enters a date: this date is converted to a unix timestamp. A SELECT query is then used to find all paid invoices, the paid dates are retrieved and then also converted to timestamps. If, for a row, the paid date's timestamp is less than the user-entered date's timestamp the relevant details are converted to .CSV format and written to a file. This file is named by the current date then saved on the client machine in the same way as the files created by dump.php. This file uses a code adapted from a php script available at <http://207.44.136.24/mysqlexport.php>.

header.php: Since every file must connect to the MySQL database, it saves on code to do this in one file (which is linked to all other files by means of PHP's *include* function). This file also contains a number of functions for populating drop down menus with data from the database: for example in projects.php the drop-down customer menu on the 'add project' form is implemented by getting all

customer names using a SELECT query and looping through them, creating options for a menu. The function to convert dates from DD/MM/YYYY to MM/DD/YYYY is also included here.

displayinvoice.php: This file displays an invoice in a form in which it can be printed and sent to a customer. Input to this page comes from a GET request, called by clicking the 'display invoice' button on an invoice returned by *displayinvoice.php*.

newsupplier.php: This adds a new supplier for an existing stock item. The user arrives here by clicking 'new supplier' for an item returned by *displaystock.php* using the GET method to send the stock_id of the item. The page uses this stock_id in a series of queries that return all suppliers that do not stock this item and adds them to a drop down menu. The user can choose a supplier from the menu. The supp_id of this supplier along with the stock_id of the item will be inserted into the supply_stock table by means of an INSERT query.

addstock.php and removestock.php: These update stock values using an UPDATE query. The user enters the amount which is posted back to the same page to create the query (after validation). The user arrives at this page from *displaystock.php* using the GET method to send the stock_id of the item being adjusted.

newloc.php: This adds a new location to the list of locations that can be chosen for a project, using an INSERT query. This required an extra table 'locations' in the database. The user arrives here by clicking the 'New' button next to the list of locations on the 'Add Project' form.

warnstock.php and warninv.php: These open in a separate window and warn users of any stock shortages or invoices that have not been paid for a certain time. These files have a function that displays the offending items.

projtoinvoice.php: This creates an invoice for a project using the project's name and details as the new invoice's name and details. The user arrives at this page by clicking 'create invoice' for a project returned by *displayinvoices.php* which uses a GET request to send the proj_id of the project. This proj_id is used in SELECT query to return the name and details of the project in a form that can be edited. The user must input a price and these details are used to create a new invoice using an INSERT query.

3.5 Illustrated example of system in use

In an attempt to better explain the workings of the system, the following diagram is shown. This diagrams illustrates the flow of variables involved in a typical case of use where a user displays information for customer 'John Smith' by entering his name in a text area, then deletes his records from the database:

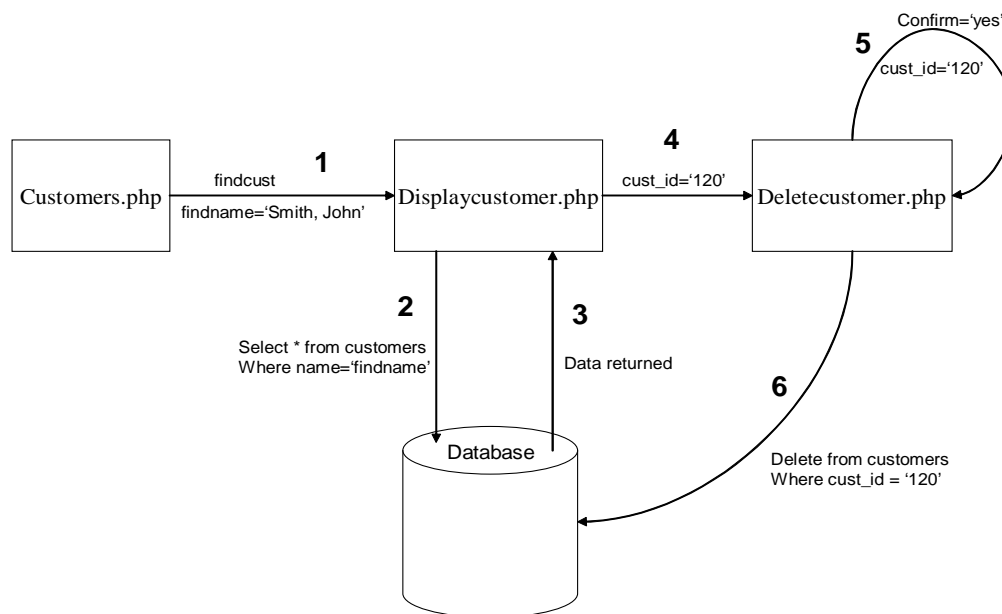


Figure 3-3 Flow of variables in the system

1. The variables *findcust* and *findname* (with value 'Smith, John') are posted to *displaycustomers.php*.
2. The presence of *findcust* causes *displaycustomers.php* to issue an SQL query to the database using the value of *findname* in the WHERE clause.
3. Data is returned and displayed, with the *cust_id* of John Smith (120) being used to build the hyperlink used for the delete button (*deletecustomer.php?cust_id=120*).
4. Clicking this delete button sends this *cust_id* to *deletecustomer.php*. *Deletecustomer.php* issues a SELECT query to get the customer name (not shown on the diagram) and ask the user to confirm deletion.
5. The user confirms by clicking a 'yes' button which is actually a hyperlink to *deletecustomers.php?confirm=yes&&cust_id=120*.
6. On receiving the confirm variable, *deletecustomer.php* issues a DELETE query using the *cust_id* value in the where clause. The record is deleted and the user is notified.

Note that in **1** variables are sent using the POST method, whereas in **4** and **5** the GET method is used.

3.6 Other Technologies used

JavaScript: JavaScript is a client side scripting language. Unlike PHP which is executed on the server, JavaScript code is downloaded and executed by the client's browser. JavaScript was used to open new windows (with specified sizes and positions), to display data, messages, and warnings. JavaScript is also commonly used to validate form data (validating it before it is submitted) however a browser with JavaScript disabled/not supported could allow invalid data to be input, so in this project all validation was carried out in PHP.

Dreamweaver MX: Dreamweaver MX is a powerful and easy to use website building tool. It was used to quickly create the prototype. While it was useful in creating the general layout of the pages, the HTML code it generated required much fine tuning by hand.

GIMP: GIMP (GNU Image Manipulation Program): Is a freely distributed image manipulation program. It was used to create all the images (most of which were used as buttons).

GVIM: This powerful text editor was used for writing the PHP files. It recognises most popular programming and scripting languages including HTML and PHP; it uses different colours to represent functions, variables and text-strings making code clear to read.

3.7 Testing

Functionality testing: Testing was carried out extensively during implementation. Once the final system had been built, a test plan designed to capture every aspect of the systems functionality was used to ensure that the system was free of any major bugs. A portion of this test plan (that dealing with stocks and customers) is shown in Appendix D. This testing identified only one problem: an SQL error in *newsupplier.php* which was fixed immediately.

Acceptance testing: Acceptance testing is carried out by the users of the system, in order to give final approval. Normally this would be carried out with real data (Avison and Shah, 1997) but this was

impossible due to time and geographical constraints. Acceptance testing was carried out before installation by hosting a dummy-data version of the system on a remotely accessible web server so the users could test the system from their web browser at home

Acceptance testing was carried out by having the users perform a number of typical tasks, reporting any problems they discovered. Tasks chosen should attempt to expose any flaws in the system, by having the users carry out operations that could lead to problems as well as trying to reflect typical cases of the system's use (Rubin, 1994). The tasks chosen attempted to represent a wide range of typical tasks, as well as places where there was usability concerns. For example task 8 involves editing stock, one of the features of the system likely to be used most commonly. Task 4 involves creating an invoice for a project, perhaps it is not obvious that this is achieved from the projects page rather than the invoices page; tasks such as this were chosen to attempt to expose any usability flaws.

A method of testing which has been commonly used in evaluating the usability of websites (such as in Spool, 1999) was also employed, whereby users were asked number of factual questions regarding data in the system. This tested the ease with which users could retrieve data from the system.

The tasks and questions used are shown in Appendix E.

To really test the usability of the system in terms of recognition rather than recall (see 2.8.1), users were not given a copy of the user manual when carrying out the list of tasks and answering the questions. With large-scale projects, feedback is collected from testers through questionnaires – however since only two users were testing the system in this project, they were simply invited to comment on anything they found difficult. The findings of the usability tests are discussed in 4.2.5.

3.8 Installation

Database: Phpdev was installed on the office machine, and phpmyadmin was used to create a new database and a new user with a password for accessing this database.

Phpfiles: Phpdev creates a folder 'www' where all web pages are stored by default. For example when the server is running on the local host the file c:\phpdev\www\index.php can be accessed by pointing a web browser to <http://localhost/index.php>. The PHP files were copied into this directory.

header.php had to be altered to change the username and password for connecting to the new database.

Data Entry: Real customer, supplier, stock and project data was entered into the system. As a customer has to be chosen for each project added, customer details had to be entered into the system before project detail. Similarly, supplier details had to be added before stock items. Invoice data has not yet been added due to a problem with the invoice functionality as discussed in chapter 4, details of old invoices will not be added to the system, as users wanted the system to only store invoices from the installation date onwards.

Running the system: A Shortcut to <http://localhost.php> was placed on desktop and in a folder in the start menu. Since Apache has to be running, the computer was configured to run Apache at start-up. For security Apache was configured to not accept external connections.

CHAPTER FOUR: EVALUATION

Overall the project can be considered a success in that the system met all functional requirements, was easy to use, and went some way to facilitating the possible future expansions: remote access and integration of the current accounts system. However the project could have been improved in several ways. This chapter provides a detailed evaluation of all stages of the project.

4.1 Methodology

4.1.1 Choice of Methodology

Using an iterative process helped break down the project into manageable sections and ensure it was completed on time. However, not using any formal methodology for actually writing the code led to problems that only became apparent when implementation started. While the database and user interface had been designed in detail, the structure of the code itself had not. This meant that code was written in a fairly haphazard way, simply implementing a new file when it was needed. Often time was wasted having to redesign certain aspects of the system as it became apparent that one approach was not going to be successful. This lack of organisation also attributed to the final code being fairly messy which made debugging difficult and will hinder any future expansions of the system. It was somewhat naïve to assume the database could be created and just ‘plugged in’ to an interface without requiring detailed planning of code.

4.2 Design

4.2.1 Analysis of the Current System

The initial visit to Gibilaro design was delayed by two weeks. In order to push on with the project several phone calls were made to try and get some idea of the running of the current system. However little useful information could be obtained through these phone calls and it was only after observation of the system and a face-to-face interview with Karen Gibilaro (see Appendix C) that the analysis could be carried out in detail. While this did cause delay, the lesson was learnt and two subsequent visits were carried out.

In order to determine how an invoice should be displayed by the system, some example invoices were examined. In fact the entire section of the system that dealt with invoices was built around these documents. This led to a problem in that in reality, invoices exist that are for more than one item and the system allowed for only one item in an invoice. Addressing this oversight requires a change to the database as well to all code dealing with invoices, this change will take several days to implement – a completion date of 01/05/2004 has been agreed upon. This revised database design has invoices composed of one or more invoice items as shown on the following ER diagram:

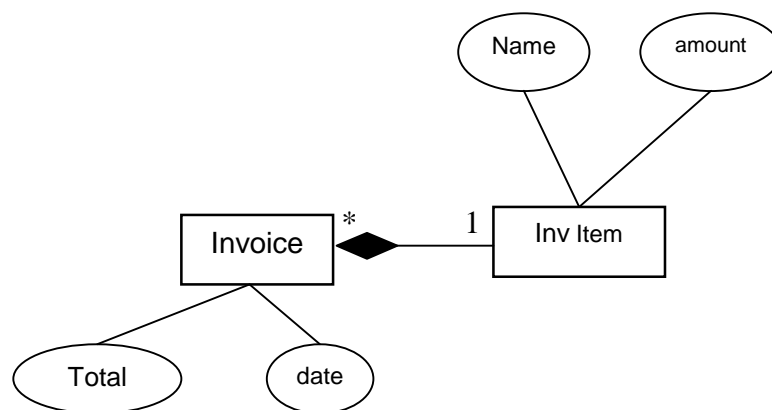


Figure 4-1 Revised ER Model for invoices.

Here the analysis of the current system undertaken was not detailed enough in respect to invoices and led to a serious oversight. Perhaps more face to face meetings with users would have prevented this from happening. Until this problem is addressed the users will not be using the invoice functionality of the system.

4.2.2 Functional Requirements

After extensive testing (described in 3.7) it can be confirmed that all of functional requirements detailed in 2.2 are met by the system. Along with some extra features that were added during implementation as they were considered to be beneficial to user (e.g. the ability to email a customer, or view a supplier website).

The difficulty was not with going from a set of user requirements to a working system, but from the idea of a new system to a set of user requirements as the users only knew what they wanted in general terms. Input from the users, a detailed analysis of the existing system, and use of a prototype led to a set of requirements both myself and the users were happy with. However there was still an oversight

(the problem of invoices being made of multiple items). Choosing the right level of detail for functional requirements is a delicate process, as explained in **2.2**. Perhaps had the requirements been more detailed, the problem would have been noticed by myself or the users.

4.2.3 Choice of technology

There were no great problems or limitations found with the choice of technology for the system. In fact PHP was found to be even more powerful and flexible than first thought. The GET method was used extensively to implement buttons, using data from the database to create hyperlinks but this approach was not discovered until after the technology had been decided on. PHP has a similar syntax to many popular programming languages so learning it was a fairly straightforward process – especially with the help of reference books such as Ullman L, (2003).

There is a large amount of online help available for PHP and MySQL. Developer forums such as devarticles(2003) were used fairly regularly, and some of the code used in the system is adapted from scripts found online (details are in **3.3.7**).

The only real difficulties were with debugging. PHP error messages tend to be fairly vague and often point to the wrong line of code. Also when using a number of different technologies (PHP, MySQL, HTML, and JavaScript) it is hard to determine where the problem is when output is not as expected. Hours spent tracking down simple syntax errors could have been put to more productive use: e.g. tidying up code, or adding additional functionality to the system.

A further advantage was discovered with separating the database from the user interface in that some modifications can be made to resolve the problems of browser compatibility discussed in **4.2.3** without causing any disruption to the users. This is because no changes need to be made to the database, the modified interface will simply replace the existing one. In effect, separating the user interface from the database has facilitated maintenance of the system.

Use of Dreamweaver MX was intended to speed up implementation; however, the HTML code generated required a lot of adjustment. Since the HTML code required was fairly simple, it would probably have been quicker to not use Dreamweaver at all.

4.2.4 Database

Apart from the oversight described in **4.1.2** the initial database design was solid and did not need to be changed at all during implementation – although extra tables for usernames and password, and

storage locations were added. Correct use of ER modelling techniques and mapping the model to a relational database led to a scheme that was in Boyce-Codd normal form, so any serious anomalies in the data would never arise.

4.2.5 User Interface

Usability was key in this project due to the inexperience of the users. Overall the usability tests were carried out with ease, users found correct answers to all the questions, and were able to complete all the tasks. Only two minor problems were discovered.

One problem encountered was in task 8. This task involved adding a new stock item for a supplier, displaying it and updating it. The problem was that the new supplier did not appear in the drop-down menu, since the page has to be refreshed to reload supplier names from the database. The users did not realise the page had to be refreshed, so resorted to pressing the 'display all suppliers' button to get round the problem. This problem was address by having the system prompt users to refresh the main page whenever adding data that is used to populate a menu on that page.

Another issue with the user interface was discovered during usability testing. The problem occurred when users displayed data then did not click the close button – they just clicked somewhere off the window causing it to be minimised. Users would then try and to display some other data and it would appear nothing had happened since a new window was already opened. This did not turn out to be a major problem as users soon realised what was happening, and would check for a minimised window if data did not appear.

4.2.6 The Prototype

It was hoped that the prototype would play a major role in the design of the interface, and in requirements capture. While user feedback from the interface did lead to some design changes (see 2.6) none of these were particularly significant. Although care was taken in picking the level of detail of the prototype (see 2.6), perhaps an even less detailed prototype would have been more effective in discovering user requirements.

4.3 Implementation

4.3.1 Coding Issues

At one point in the implementation a problem arose regarding SQL syntax. *newsupplier.php* has to find a list of all suppliers who do not supply a particular stock item. While this sounds trivial, it was complicated by the many-many relationship between stocks and suppliers. It would be possible to find this information with a single SQL query using a set difference, but poor understanding of advanced SQL syntax meant that this approach had to be abandoned. In the end, multiple queries were used to create two PHP arrays (one with all suppliers, one with all suppliers for an item), subtract them using the PHP *array_diff* function, then use the new array to build a drop down menu of suppliers - some 50 lines of code! As well as being unnecessarily complicated, this code is clearly inefficient as it involves multiple database accesses where only one is needed.

There are other areas of implementation where code could have been simplified with better use of .SQL. Over-complicated code further hindered the already difficult debugging process.

4.3.2 Remote Access

The system was designed to be easily expandable for remote access. A password system is in place and most of the functionality of the system works remotely (see 2.3.2). However to really realise remote access a number of changes will have to be made.

The backup function currently requires the backup file to be on the server. The system will have to be modified to upload a file to the server, and then perform the backup operation. Linking to a project's documents could be achieved by hosting them on the web server; this would involve some rearrangement of the company's electronic folders so it would obviously have to be discussed with users prior to implementation.

Even though there is a password system, this is not enough to ensure that the system would be completely secure when available for remote access. Some form of encryption would have to be used to ensure that passwords could not be stolen in transit. HTTPS (created by Netscape), which uses a secure socket layer below HTTP that encrypts and decrypts client requests and server replies, would be a suitable choice. Setting up HTTPS would require an additional Apache module to be installed, on Windows this is fairly straightforward and does not require Apache to be recompiled (JM Solutions, 2003).

The real problem with remote access is the issue of browser compatibility. Testing the system on Mozilla was disappointing to say the least: images seemed to randomly disappear and scroll bars did not work at all. The cause of the former problem (the direction of slashes on image tags) has now been identified but as of yet no cause for the missing scroll bars has been found. Internet Explorer was chosen as the browser that would be used for the system (see **2.5.6**), so when building the system it was the only browser used for testing. This was clearly unwise, as remote access will mean it should be functional on all browsers. More importantly, if the users decided to change their browser of choice they could run into problems. For this reason the user interface will soon be replaced with a cross-browser compatible one. This would cause no disruption to the users as explained in **4.2.3**.

It was intended that the system should be easily expandable to be accessed remotely, but at this stage a fair amount of work will still have to be done to achieve this.

4.4 Overall Success of the Project

Overall the project can certainly be described as a success. Using an appropriate methodology, and choice of technology led to a system that was finished on time with all functional requirements met. **There were problems however**, the main criticism being that lack of detailed planning of code structure and limited knowledge of SQL led to code that was untidy and thus hard to debug and expand.

Usability was always a key concern, and the interface was designed to be as easy to use as possible, following usability principles as discussed in chapter 2. The results of the usability tests were extremely positive – users were able to carry out all tasks and answer all questions correctly with no major problems.

Although there was an oversight with regards to invoices, this will be addressed promptly and the invoice section of the system should be fully functional by 01/05/2004. The browser compatibility problems will also be resolved in the very near future with no disruption to the users,

The overall success of the project can really only be judged by how the system performs in the future. So far, feedback from the users has been encouraging, but only time will tell if the system fully realises its goals in solving the information management problems faced by Gibilaro design.

Bibliography

1phpstreet, 2004, *Script to dump mysql database to file*: -

URL:<http://www.1phpstreet.com/vb/scripts/ShowCode.asp?txtCodeId=772&lngWId=8>[12th March 2004]

Augerou C and Cornford T, (1998), *Developing Information Systems: Concepts Issues and Practice 2nd edition*, Houndmills: Palgrave.

Avison D.E and Shah H.U , (1997), *The information systems development life cycle: a first course in information systems*, London: McGraw-Hill.

BCP, *Stock Control Software - Supply Chain Software: UK Retail & Wholesale Distribution Systems from BCP*, URL:http://www.bcpsoftware.com/solutions/stock_control.php[2nd December 2003]

Chen P, (1976), 'The Entity Relationship Model - Towards a Unified View of Data', ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 9-36.

Codd E, (1970), *A Relational Model for Large Shared Data Banks*, CACM, 13:6, June 1970.

Codd E, (1972), *Further Normalisation of the of the Data Base Relational Model*, in Rustin(1972).

Devarticles.com, (2004), *Quick mysql dump from file using php*, The forums of devarticles.com (<http://forums/devarticles.com>),
URL:<http://forums.devarticles.com/archive/t-6007>[20 March 2004]

Elmarsri R and Navathe S B, (2000), *Fundamentals of database systems 3rd edition*, Reading:Addison-Wesley.

Firepages.com.au, 2001, URL:<http://www.firepages.com.au>[2nd January 2004].

Hughes B and Cotterell M, (2003), *Software Project Management 3rd Edition*, London: McGraw-Hill.

JM Solutions, *mod_ssl and OpenSSL*, URL:<http://www.jm-solutions.com/OpenSSL/Setup/setup01.php>[22nd April 2004].

Johnson G, (2002), *A Stock Control System for a record shop*.

Kamthan P,(1999), *Apache at your web service*,

URL:<http://tech.irt.org/articles/js177/#advantages>[3rd January 2004]

Netcraft – URL:<http://www.netcraft.com>[21 January 2004].

Nielson J, 2004, *Ten Usability Heuristics*

URL:http://www.useit.com/papers/heuristic/heuristic_list.html[1st February 2004].

Number One Systems, *Stockit*, URL:<http://www.numberone.com/stockit.htm>[29th October 2003].

Ritche C, (2002), *Relational Database Principles 2nd edition*, London: Continuum.

Rosson M B and Carrol J M, (2002), *Usability engineering: Scenario based development of human-computer interaction*. San Francisco: Morgan Kaufmann.

Rubin J, (1994), *Handbook of Usability Testing: How to plan, design, and conduct effective tests*, NYL: John Wiley & Sons Inc.

Rustin R, (1972), *Data Base Systems*, London: Prentice Hall.

Spool J M ... [et al.], 1999, *Web site usability: a designer's guide*, San Francisco, Calif.: Morgan Kaufmann, London: Taylor & Francis.

Stobart, S, (1996), *Essential PHP fast : building dynamic Web sites with MySQL*, London: Springer,

Ullman L, (2003), *PHP and MySQL for Dynamic Web sites: A visual QuickPro guide*. Berkely: Peachpit Press.

Wasserman A L and Shewmake D T, (1982), *Rapid prototyping of interactive information systems*. Software Engineering Notes, IEEE, Volume 7, Number 5, 171-180.

APPENDIX A: PERSONAL REFLECTION

This final year project has been by far the most challenging undertaking of my university life. The pressure of producing a system for a real company, together with the fact that it counts considerably in the final degree assessment, made the experience stressful at times. In the end, however, the project has turned out a success, and I come away from it with a great sense of personal achievement in having created a system which will actually be used. The exercise has also given me more confidence in my programming ability and helped me develop organisational and time-management skills.

Using a methodology that involved splitting the project up into a number of stages has rendered the whole process more manageable. The temptation was to rush in and start coding straight away. An early decision, however, was to delay this particular stage until the overall design strategy and general details had been practically finalised. Implementation is, after all, only one phase of a software project, and best left until a very clear idea of exactly what is required has been formulated. For this reason, this phase was not started until the end of the first semester (12/03/2004). While it appeared daunting to start so late, the fact that the system had already been designed made the actual implementation a relatively straightforward procedure. This is not to say it was easy. There were many late nights spent staring at code in exasperation, trying to spot syntax errors or wondering why something was not working the way it should; but this, of course, was always to be expected. The only serious implementation problems I experienced were in trying to integrate other peoples' scripts (found on the Internet) with my own code. I soon came to realise how essential it is when attempting this to understand fully the third party code: failure to do so can lead to countless problems.

With hindsight, it would probably have been better to have left the coding even later – to after having spent even more time on the detailed design of the structure and on familiarising myself further with the technology. This could well have led to a simplification of the code in some instances and alleviated some of the problems of messiness discussed in the evaluation section of this report.

A number of important lessons have been learned. All modern books on software project management will tell you that effective communication with the prospective users is of the utmost importance. The problems I had with finalising a set of requirements, and the huge advantages I found with detailed face-to-face discussions over telephone conversations show that this could not be truer. I have also learnt to make effective use of resources, discovering that, while the Internet did prove useful for

finding scripts and providing help from online forums, the most useful information was to be found in print.

I initially considered leaving the write-up till fairly late, just making notes as I went along. However, midway through the first semester, I found myself with such a mass of handwritten material that I decided to start working on it earlier. This proved to be a good decision; and, come the Easter holidays, writing-up consisted in the main of simply putting things together, filling in some gaps, and re-writing sections I was unhappy with.

The above observations may be summarised in terms of advice I would give to anyone undertaking a similar project. First, try to start the write-up as soon as possible; those leaving this till the Easter break are likely to find themselves overwhelmed and to lose out on valuable revision time. Secondly, do not feel pressured to start coding early on; if your design is solid and you understand the technology you are using, the implementation will be much more straightforward. In short, do not start doing something until you know exactly what it is you want to do! Finally, try not to become overwhelmed: there is a considerable amount of work to do, but breaking it down into a number of stages, and working consistently throughout the year, will help to keep the pressure off.

APPENDIX B: KEY STAGES IN THE PROJECT

01/11/2003 - Start of analysis stage: Research begun into software methodologies. Some past student projects examined. Existing systems such as stockit examined.

15/11/2003 – Interview and Observation: Spent two days observing the running of the company. Had an interview with Karen Gibilaro (see Appendix C) and interviewed all users with regard to their computer skills.

18/11/2003 – User Requirements: Began work on a set of functional user requirements for the new system.

08/12/2003 – Requirements Meeting: Meeting with Karen and Jesse Gibilaro to discuss functional requirements.

03/01/2004 – Start of Design Phase: Designed the database, decided on a technology, and began implementing the prototype.

06/01/2004 - Demonstration of Prototype: Prototype demonstrated to Karen and Jesse Gibilaro, leading to some design changes being made.

12/03/2004 - Start of implementation: Database and interface (expansion of the prototype) worked on alternately.

20/04/2004 - Testing: Test plan written and used to test system, usability test carried out by users.

30/04/2005 – Installation: System Installed at Gibilaro Design.

APPENDIX C: INTERVIEW WITH KAREN GIBILARO

Karen Gibilaro is the office manager at Gibilaro design. She is responsible for much of the company's information management. The interview was carried out on: 15/11/2003

This is not an actual transcript, it has been written from notes taken during the interview.

How many different stock items do you have?

There are a great many different stock items.

Metal stock - mild steel, bright steel, brass, bronze, stainless steel and some odds and ends of aluminium and copper.

Of the first five listed, we have each in bars (round/square/rect), plates (at least three different thicknesses), heavy solid section and various hollow sections (round and square).

Mouldings - maybe 15 different types

Screws, bolts, nuts, etc - 75 different types

Ball bearings - 5 different sizes (steel and brass).

Castings - 40 different types

Cast iron - two different types and two different sizes.

Plus other bits and pieces.

How do you record stock information?

At present we have no formal method for recording stock information. Some stock levels are recorded on paper as reminders.

Do you have regular stock checks to determine when things are running out?

More commonly used stock items are checked from time to time to ensure they are not running out. Stock checks on other items are sometimes carried out but not regularly.

How is stock stored?

Various different places, we have a metal rack for metal, various containers for other stock.

How many different Suppliers do you have?

There are around 20 suppliers who are used regularly.

How do you hold supplier information?

A list of names and phone numbers is held on a word document. Invoices from suppliers are held as documents too for accounts purposes. These accounts records go back 6 years.

Do you have any suppliers making regular deliveries without them having to be contacted?

No, as what we produce is made to order the stock we need is always changing, although some suppliers are used a lot more regularly than others.

How do you order?

Most of our orders are done by fax, sometimes they are done by phone depending on the supplier and who is ordering. A couple of times we have used the internet.

Do you deal directly with customers or do jobs come through an agent?

Our jobs come through dealers and architects etc. We don't normally deal with the customers. Payments too, come through the agencies but occasionally direct from customers. Sometimes we deal with the same customer through many multiple agencies!

NOTE: From here on 'dealers' refers to anyone supplying jobs to the company.

How many dealers do you have?

Between 10-15 dealers. Most of these regularly give us work.

What information do you hold on dealers and customers?

Again we have lists of names and numbers on word documents
All invoices are held for accounts purposes.

How far back do your records go?

Accounts information goes back 6 years for legal reasons.

What type of info do you hold on past jobs?

Each job has a folder (ring binder type) with all relevant correspondence (invoices*?* for orders to do with the job). Smaller jobs have a plastic file.
All kept in a filing cabinet.

How do you identify a job?

Normally by the name or address of the dealer.

Who are your current employees?

Jesse Gibilaro: Registered as the sole trader.

Karen Gibilaro: Office manager

Sam Smith: Assistant Technician.

We also have staff temporary staff who come in when needed

How exactly do you store you information for accounts?

We use four excel spread sheets.

Expenses(all costs), Receipts (all income)

Summary and VAT return (generated automatically).

With the way things are done at present, what problems are you having?

One of the main problems is stock. From time to time there will be a panic when we run out of a certain stock item. As some of our suppliers can take weeks to deliver – this can be a big problem. Our main way round this is to order more stocks than needed.

All though I have a clear idea of how everything works, when I'm not there, other employees often run into problems (for example they may not know which supplier to use for a particular stock item). The set-up could be better organised to make it easier for other employees to understand, especially when I'm away. As we are expanding it is likely we will be taking on new employees and it will be difficult for them to quickly learn how everything works – this is a problem with temporary staff with whom we can't afford to waste time.

Sometimes we want to look at a over a past job or a past invoice – we may want to keep our prices in line for a particular supplier or we may want plans from an older job. This can take ages as it means looking for the folder or the invoice document.

When creating invoices, I sometimes get the number of the invoice wrong as the only way I can find it is looking at the last one.

What exactly is stored with jobs (invoices etc.)

Not invoices, just correspondence and job plans etc.

How do you distinguish jobs if you store by dealer name/address, and is dealer name/address written on the folder.

Some have the dealer name, others have the address, some have a description.

How many jobs per year, how long do they take (range)

Around 15-20 major jobs and quite a lot of minor ones.

Where do you store invoices from suppliers?

In a ring binder stored by date.

Do you print out copies of name, address, number lists?

Yes we have two phones that are away from the office.

Do you have stock info with suppliers?

Some we do, some I just know.

APPENDIX D: PART OF TEST PLAN

The following table shows the sections of the test plan related to Stocks and suppliers.

Action	Expected Result	As Expected?
Stock		
Enter a valid stock item name and press appropriate display button.	Details for that item displayed + close button.	Yes
Press display button with no value entered.	Error message displayed.	Yes
Press display button with non-existent stock item entered.	'No items found' message displayed.	Yes
Choose a stock item from the drop-down menu and press the appropriate display button.	Details for that item displayed + close button.	Yes
Press 'display all items' button.	Details for all stock items displayed + close button.	Yes
Press the 'printable stock check form' button.	Printable details for all stock items displayed + close button.	Yes
Press the 'perform stock check button'.	All stock items displayed with updateable values + close button.	Yes
Press the 'supplier' button for a stock item.	Supplier(s) for that stock item displayed + close button + back button.	Yes
Press 'add' button for a stock item.	Jump to page for adding stock.	Yes
Enter an amount to add and press 'add'.	Stock levels updated + message displayed + close button.	Yes
Press 'remove' button for a stock item.	Jump to page for removing stock + back button	Yes
Enter an amount to remove and press 'remove'.	Stock levels updated + message displayed + close button.	Yes
Press 'new supplier' button for a stock item.	Jump to page with drop down menu containing all suppliers who do not supply the item + back button.	Yes
Choose a new supplier for a stock item and press 'add'	Supplier is now recorded as displaying that item + message displayed + close button.	Yes
Press the 'delete' button for a stock item.	Jump to page prompting 'yes' or 'no'	Yes

Press 'no' on delete stock page.	Jump back to stock item(s).	Yes
Press 'yes' on delete supplier page.	Stock item deleted + message displayed + close button.	Yes
Enter a name, choose a supplier, enter a WT and click the 'add' button on 'add stock item' form.	New stock item added with appropriate details.	Yes
Leave name blank and try to add new item.	Error message displayed	Yes
Leave WT blank and try to add new item.	Error message displayed.	Yes
Choose non-numerical WT and try to add new item.	Error message displayed.	Yes
Remove items so a stock item is below its WT. And refresh stocks page.	Warning of low stock for that item displayed.	Yes
Press 'close' button when displaying items.	Window closes.	Yes
Customers		
Enter a valid customer name and press appropriate display button.	Details for that customer displayed.	Yes
Press display button with no value entered.	Error message displayed.	Yes
Press display button with non-existent customer item entered.	'No items found' message displayed.	Yes
Press 'display all customers' button.	Details for all customers displayed	Yes
Press the 'printable list' button.	Printable details for all customers displayed.	Yes
Press 'projects' button for a customer who has at least one project.	Project(s) for that customer displayed. Along with 'back button'	Yes
Press 'projects' button for a customer who has no projects.	'no projects' message displayed.	Yes
Press 'send email' button for a customer.	New email message opens with customer's email address in the address field.	Yes
Press 'edit' button for a customer.	Jump to page with all customer's details in editable form.	Yes
Enter new values in the edit customer form and press 'update'	Customer information updated. +message displayed.	Yes
Press 'update' on edit customer form leaving out name field.	Error message displayed +back button.	Yes
...leaving out address1 field.	Error message displayed +back button.	Yes

...address2 field.	Error message displayed +back button.	Yes
...postcode	Error message displayed +back button.	Yes
...phone field.	Error message displayed +back button.	Yes
...fax field.	Error message displayed +back button.	Yes
...email	Error message displayed +back button.	Yes
Press the 'delete' button for a customer.	Jump to page prompting 'yes' or 'no'	Yes
Press 'no' on delete customer page.	Jump back to customer(s).	Yes
Press 'yes' on delete customer page.	Customer deleted + message displayed.	Yes
Enter customer details on 'add customer' form and press 'add' button.	New customer added to the database and error message displayed.	Yes
Press 'add' button on customer form leaving out the 'name' field.	Error message displayed + back button.	Yes
...leaving out address1 field.	Error message displayed + back button.	Yes
...address2 field.	Error message displayed + back button.	Yes
...postcode	Error message displayed + back button.	Yes
...phone field.	Error message displayed + back button.	Yes
...fax field.	Error message displayed + back button.	Yes

APPENDIX E: USABILITY TESTS

The following tables show the tasks and questions used to test the usability of the system.

Q No.	Question	Correct?
1	How much stock is remaining for item 'Bolt 1A'?	Yes
2	Who supplies the item 'Sheet Metal 2P'?	Yes
3	What is customer Fred Testerson's phone number?	Yes
4	Customer Jake Gibilaro has two projects. What are their names?	Yes
5	When was the invoice 'candle sticks' created?	Yes
6	And when was it paid?	Yes
7	Which invoice has the lowest price, and who is it for?	Yes
8	This customer has a project, what is it called?	Yes
9	How many days old is the oldest unpaid invoice?	Yes
10	One item is running low on stock, how much is left?	Yes

Task No.	Task	Completed?	Comments
1	Login to the system.	Yes	
2	Back up the database to a file, then restore the database from this backup.	Yes	
3	Add a new customer. Edit this customer change the phone number.	Yes	
4	Add a project for this customer. View this project and create an invoice for it for £100.	Yes	
5	Show all invoices. Display the invoice you have just created.	Yes	
6	Set this invoice to paid. Convert all paid invoices to a file that can be used by Excel.	Yes	
7	Display the details for the supplier who stocks 'Nail 1A'. Go to this supplier's website.	Yes	
8	Add a new stock item for supplier 'Terry's Tools', so that you are warned when there are 100 left. Display this item and add 200 units of stock.	Yes	Could not select it from drop-down menu. Had to display all items.
9	Perform a stock check and change the amount of this item to 150.	Yes	
10	Add a new supplier for this item.	Yes	

APPENDIX F: THE USER MANUAL

Contents

1. Navigation	1
1.1 Close and back buttons	1
2. Starting Page	1
2.1 Backup and Restore	1
3. Stock	2
3.1 Warnings	2
3.2 Adding and removing stock	2
3.3 Stock Checking	3
3.4 Adding a new stock item	4
4. Customers	4
4.1 Displaying Customers	4
4.2 Printable list	6
4.3 Adding a customer	6
5. Suppliers	7
5.1 Displaying Suppliers	7
5.2 Adding a supplier	8
6 Projects	9
6.1 Displaying Projects	9
6.2 Adding a Project	10
7. Invoices	11
7.1 Warnings	11
7.2 Displaying Invoices	12
7.3 Adding an Invoice	12
7.4 Paid invoices	13

1. Navigation

The system is divided into 6 pages. To go to any page use the menu at the top of the screen:

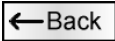


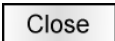
The question mark on the right hand side will always bring you back to the starting page.

The page that you are currently on will appear darker in the menu. For example:




1.1 Close and back buttons

Clicking  will always take you back to the page that you came from.


Clicking  will close the window.

2. Starting Page

This page gives you information on what the other pages of the system do.

For more detailed information it is possible to bring up this user manual by clicking 

2.1 Backup and Restore

From this page you can also back up the database. To do this click 

A message will pop up – click save and choose a location to save the file then click save again to save it (The file will be named by the current date e.g. (11-12-2004.sql).

To restore the database from a previous backup, click **Restore**. A window will pop up, click browse to choose a .sql file to backup the database from, click open and then restore to restore the database.

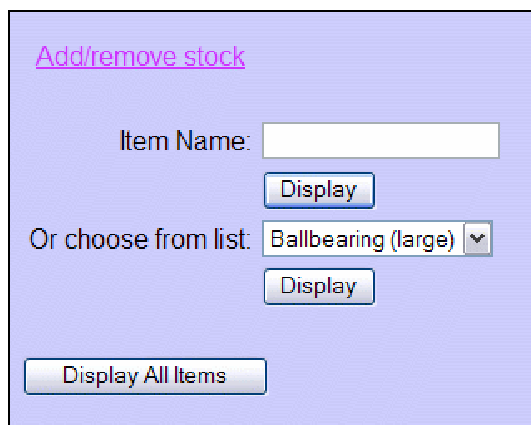
3. Stock

3.1 Warnings

When arriving at the stock page, the system will provide warnings of any stock items that are running low.

3.2 Adding and removing stock

The form at the top left of the stock page is used for displaying stock items and adding and removing stock:



Add/remove stock

Item Name:

Or choose from list:

To display details of a stock item either enter the item name in the Item Name box, or use the drop down menu and press the corresponding display button. To display details of all items click 'Display all items'.

NOTE: If using the item box make sure the name is entered exactly as it is stored in the database. This applies to any case where text is entered in to the system.

Items will be displayed as follows:

name	quantity			
Ballbearing (large)	25	Add	Supplier	Delete
		Remove	New Supplier	
Ballbearing (small)	99	Add	Supplier	Delete
		Remove	New Supplier	

To add or remove items from stock: press **Add** or **Remove** , enter the number and press the appropriate button.

NOTE: Stock levels cannot fall below zero (if you have 100 and remove 200, 0 will be recorded).

Click **Supplier** to display details of all the suppliers of the stock item (see **Suppliers** section). To add a new supplier for the stock item click **New Supplier** , you will be able to choose a new supplier for the item from a list – click add to add this supplier.

To delete a stock item click **Delete** .

3.3 Stock Checking

The form at the bottom left of the page is for stock checking:

Stock Check

Printable Stock Check Form

Perform Stock Check

Clicking the Printable Stock Check form will pop up a list of all stock items along with a space where you can enter the stock levels by hand once the list is printed.

To print the list, right click the screen and click print.

To carry out a stock check of all items click the Perform Stock Check Button. This will bring up a list of all stock items as follows:

name	quantity	Enter new quantity
Ballbearing (large)	25	<input type="text" value="25"/>
Ballbearing (small)	99	<input type="text" value="99"/>

Enter the new quantities in the boxes and then click update to update all stock items.

It is also possible to bring up supplier details from here by clicking [Supplier](#)

3.4 Adding a new stock item

The form at the right of the page is used to add a new stock item to the database.

[Add a new stock item](#)

Item Name*:

Supplier*: ▼

(additional suppliers can be added later)

Warn when only* are left.

(*required fields)

Enter the name of the new item and choose a supplier from the list (additional suppliers can be added later as explained in *Adding and removing stock* above.

The system will warn you if the stock levels fall below a certain value: enter this value.

Click Add to add the details of this new stock item to the database.

4. Customers

4.1 Displaying Customers

The form on the left of the page is used to display customer information:

Display Customers

Enter Name:

Or select from list:

Either enter a name or select it from the list and press the appropriate button to bring up a list of customers.

Click Display All Customers to bring up a list of all customers.

Customers will be displayed as follows:

Name	address	numbers	email		
Gibilaro, Jake	12 Burchett Place Leeds LS6 2LN	tel:07782 308 168 fax:0113 112 212	woodhousemassif@ntlworld.com	<input type="button" value="Projects"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Offerton, James	12 Ashton Road London SW12 PGT	tel:020 345 8972 fax:020 183 3897	james.offerton@ntlworld.com	<input type="button" value="Projects"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Click to send an email to the customer.

Clicking will bring up a list of all customer projects.

Clicking will take you to a window where you can change any of the customers details – press Update when you are done. To delete a customer from the database click .

NOTE: Deleting a customer will also cause all projects and invoices for that customer to be deleted.

4.2 Printable list

The Printable list button brings up a list of customers with just their names and numbers. Right click the screen and click print to print this list.

4.3 Adding a customer

The form on the right of the page is for adding a customer to the database:

[Add a customer](#)

Name*:

Address1*:

Address2:

Post Code*:

Telephone*:

Fax:

Email:

(*required fields)

Fill in the details of the new customer (fields marked with a * must be filled in or you will receive an error message). Then click Add to add the customer.

5. Suppliers

5.1 Displaying Suppliers

The form for displaying suppliers is on the left of the screen:

Display Suppliers

Enter Name:

Or select from list:

Enter Stock item:

Or select from list:

You can display details for a supplier by entering the name of the supplier to display or choosing it from the drop down menu and pressing the appropriate button.

It is also possible to find all suppliers for a particular stock item. To do this Either enter the name of the stock item or select it from the list and press the appropriate button.

To display all suppliers click Display All Suppliers.

Suppliers will be displayed as follows:

Name	address	numbers	email	website	
Barry's Bolts	15 Whale drive	tel:020 7622 5765	barry@bolt.net	No	<input type="button" value="Edit"/>
	London SE12 APM	fax: 020 7652 1221	<input type="button" value="Send Email"/>	website recorded	<input type="button" value="Delete"/>
Bob's Ballbearings	12 Turkey Street	tel:020 7622 5765	bob@ballbearings.net	<input type="button" value="Website"/>	<input type="button" value="Edit"/>
	London SW3 1LA	fax: 0113 982 022	<input type="button" value="Send Email"/>		<input type="button" value="Delete"/>

To send an email to the supplier click **Send Email** .

If the supplier has a website you can click **Website** to open the supplier's website in a new browser window. To view the web address of the supplier, just place the cursor over the button without clicking it.

Clicking edit will take you to a window where you can change any of the supplier's details – press Update when you are done. To delete a supplier from the database press **Delete** .

NOTE: If deleting a supplier causes a stock item to have no suppliers, the item will also be deleted from the database.

Clicking **Edit** will take you to a window where you can change any of the supplier' – press Update when you are done.

5.2 Adding a supplier

The form on the right of the page is used for adding a supplier to the database:



Add a supplier

Name*:

Address1*:

Address2:

Post Code*:

Telephone*:

Fax:

Email:

Website:

(*required fields)


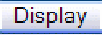
Fill in the details of the new supplier (fields marked with a * must be filled in or you will receive an error message). Then click Add to add the supplier.

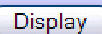
6. Projects

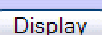
6.1 Displaying Projects


The form on the left of the page is for displaying projects:

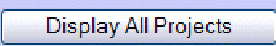
Display Projects

Name: Big fireplace 


Enter Customer:


Choose Customer: Another, One 


Choose Year: 1998 











Choose the name of the project or enter the name of a customer and press the appropriate button to display projects. It is also possible to choose a customer name from a list.


To show all projects for a particular year, choose the year from the list and click the appropriate button.


To display all projects click Display all Projects.



Projects will be displayed as follows:

Name	Customer	Description	Date	Folder	Storage	
12 candlesticks	Testerson, Fred	Made 11 novelty candlesticks and an egg cup	02/01/2003		upstairs top shelf	 
Big fireplace	Gibilaro, Jake	Installed a great big fireplace in the living room.	01/02/2004		downstairs main filing cabinet	 

Clicking  will take you too a window where you can edit the name and description and enter a price to create an invoice with the current date (see Invoices for more on invoices).

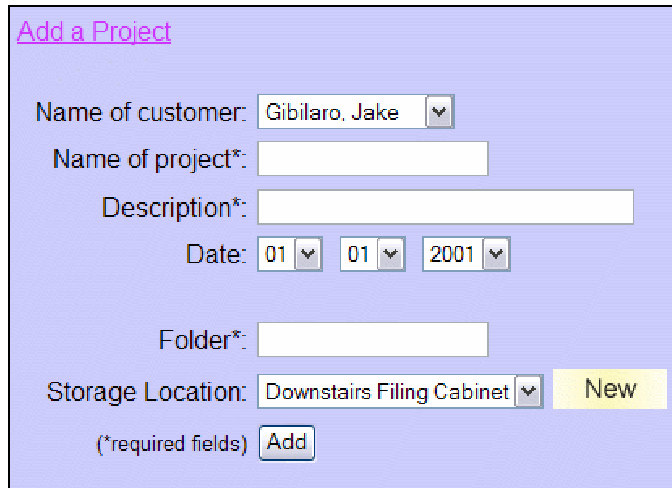
Clicking  underneath the customer name will take you to that customer's details (it will be possible to return to the projects by clicking the back button).

Clicking on the  will take you to the projects folder stored on the computer.

Clicking  will take you to a window where you can change any of the project's details – press Update when you are done. To delete a project from the database press .

6.2 Adding a Project

The form on the right of the page is used to add a project to the database:



Add a Project

Name of customer: Gibilaro, Jake ▼

Name of project*:

Description*:

Date: 01 ▼ 01 ▼ 2001 ▼

Folder*:

Storage Location: Downstairs Filing Cabinet ▼ **New**

(*required fields) **Add**

Choose a customer from the list then enter a name and description for the project.

Choose a date using the menus and enter the computer folder for the projects files (e.g.

c:\projects\myproject). Choose the location of the projects paper files from the menu. Click Add to add the new project to the database.

To add more locations to the list locations click: 

7. Invoices

7.1 Warnings

When arriving at the invoice page, the system will provide warnings of invoices that are unpaid after 30 days, and the number of days old each one is.

7.2 Displaying Invoices

The form for displaying invoices is on the left of the page:

Display Invoices

Enter Customer Name:

Or select from list:

Choose Year:

To display invoices for a customer, enter the customer name or choose it from the menu and press the appropriate button. You can also display all invoices for a particular year by choosing a year from the menu and pressing the corresponding display button.

Invoices are displayed as follows:

name	customer	date	amount (excl. VAT)			
Towel Rails	Smith, John <input type="button" value="View"/>	02/01/2004	75	<input type="button" value="Display"/>	<input type="button" value="Paid"/> <input type="button" value="Renew"/>	<input type="button" value="Delete"/>
Candle Sticks	Offerton, James <input type="button" value="View"/>	25/03/2004	20	<input type="button" value="Display"/>	Paid on 9/4/2004	<input type="button" value="Delete"/>

NOTE: Unpaid invoices are have Paid and renew Buttons whereas paid invoices show the date they were paid on.

To view details of a customer who an invoice is for, click **View** under the customer name.

To display an invoice in full so that it can be printed and sent to the customer, click **Display**. This will bring up the full invoice in a separate window. To print it, right click the screen and click print.

To set an invoice as paid, click **Paid**. This will record the invoice as paid on the current date.

Clicking **Renew** will display an invoice with the date set to the current date. This will not change the invoice date stored in the database.

To delete an invoice click **Delete**.

7.3 Adding an Invoice

The form at the top right of the screen is used to add an invoice to the database:

Add an Invoice

Name of customer: Another, One ▼

Name of invoice*:

Description*:

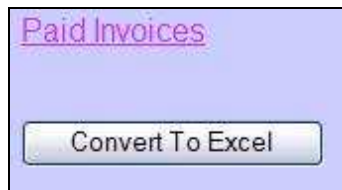
Price(£)* excl.VAT

(*required fields)

Choose a customer name and enter a name, description and price for the invoice and click add.

7.4 Paid invoices

The form for paid invoices is located at the bottom right of the screen:



Clicking 'Open in Excel' will allow you to copy paid invoices to a format that can be opened in excel. A message will pop up, Enter a date as requested, all paid invoices since this date will be converted to .CSV format which can be opened in Excel. A message will pop up, click 'save' and choose a location to save the file to.

